Goby v2

Generated by Doxygen 1.8.1.2

Wed Mar 26 2014 01:42:06

Contents

1 Goby Underwater Autonomy Project 1

1.1 Resources . . . 1
1.2 Developer manual . . . 1
1.3 Publications . . . 1
1.4 Download and Install Goby . . . 1
1.5 Building Examples . . . 2
1.6 Authors . . . 2

2 goby-acomms: An overview of Acoustic Communications Library 2

2.1 Quick Start . . . 2
2.1.1 Acoustic Communications are slow . . . 3
2.1.2 Efficiency to make messages small is good . . . 4
2.1.3 Total throughput unrealistic: prioritize data . . . 4
2.1.4 Despite all this, simplicity is good . . . 4
2.1.5 Component model . . . 4
2.2 dccl: Encoding and decoding . . . 5
2.3 queue: Priority based message queuing . . . 5
2.4 modemdriver: Modem driver . . . 6
2.5 amac: Medium Access Control (MAC) . . . 6
2.6 Software concepts used in goby-acomms . . . 6
2.6.1 Signal / Slot model for asynchronous events . . . 6
2.6.2 Google Protocol Buffers . . . 7
2.7 UML models . . . 8

3 goby-acomms: DCCL (Dynamic Compact Control Language) 9

3.1 Designing a message . . . 9
3.2 DCCL Protobuf Options . . . 12
3.3 Interacting with the DCCLCodec . . . 13
3.4 Encryption . . . 14
3.5 Example messages . . . 14
3.5.1 Minimal functional DCCL message . . . 14
3.5.2 Two Message example . . . 15
3.5.3 DCCL Test2 showing an embedded message encoded by a custom (non-default) codec . . . 18
3.5.4 DCCL Test3 . . . 18
3.5.5 DCCL Test4 . . . 19
3.5.6 DCCL Test5 . . . 19
3.5.7 DCCL Test6 . . . 20
3.5.8 DCCL Test7 . . . 20
3.5.9 DCCL Test8 . . . 21

3.5.10 DCCL Test9 . . . 21

4 goby-acomms: queue (Message Priority Queuing) 22

4.1 Understanding dynamic priority queuing . . . 22
4.2 Queuing Protobuf Options . . . 23
4.3 Interacting with the QueueManager . . . 26
4.3.1 Instantiate and configure . . . 26
4.3.2 Signals and (application layer) slots . . . 26
4.3.3 Operation . . . 27
4.4 Example messages . . . 27
4.4.1 Minimal functional DCCL / Queue message . . . 27
4.4.2 Test1 . . . 27
4.4.3 Test2, Test3, Test4 . . . 28
4.4.4 Test5 . . . 28

5 goby-acomms: amac (Medium Access Control) 29

5.1 Supported MAC schemes . . . 29
5.2 Interacting with the goby::acomms::MACManager . . . 29

6 goby-acomms: modemdriver (Driver to interact with modem firmware) 31

6.1 Abstract class: ModemDriverBase . . . 31
6.1.1 Interacting with the goby::acomms::ModemDriverBase . . . 31
6.2 Protobuf Message goby::acomms::protobuf::ModemTransmission . . . 32
6.3 Writing a new driver . . . 32
6.4 WHOI Micro-Modem Driver: MMDriver . . . 36
6.4.1 Supported Functionality . . . 36
6.4.2 Micro-Modem NMEA to Goby ModemTransmission mapping . . . 36
6.4.3 Sequence diagrams for various Micro-Modem features using Goby . . . 44

7 goby-util: Overview of Utility Libraries 50

8 goby-moos: An overview of the Goby/MOOS interoperability library 53

8.1 iFrontSeat . . . 53
8.1.1 Writing a new driver for iFrontSeat . . . 53

9 Todo List 59

10 Module Index 59

10.1 Modules . . . 59

11 Namespace Index 59

11.1 Namespace List . . . 59

12 Class Index 59

12.1 Class Hierarchy . . . 59

13 Class Index **63**

13.1 Class List . . . 63

14 File Index **66**

14.1 File List . . . 66

15 Module Documentation **78**

15.1 API classes for the Dynamic Compact Control Language (includes writing custom encoders). . . . 79

15.1.1 Detailed Description . . . 79

15.2 API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver). . . . 79

15.2.1 Detailed Description . . . 80

16 Namespace Documentation **80**

16.1 goby Namespace Reference . . . 80

16.1.1 Detailed Description . . . 82

16.1.2 Function Documentation . . . 82

16.2 goby::acomms Namespace Reference . . . 82

16.2.1 Detailed Description . . . 85

16.3 goby::common Namespace Reference . . . 85

16.3.1 Detailed Description . . . 87

16.4 goby::common::tcolor Namespace Reference . . . 88

16.4.1 Detailed Description . . . 88

16.4.2 Function Documentation . . . 88

16.5 goby::pb Namespace Reference . . . 89

16.5.1 Detailed Description . . . 89

16.6 goby::transitional Namespace Reference . . . 89

16.6.1 Detailed Description . . . 91

16.6.2 Typedef Documentation . . . 91

16.6.3 Enumeration Type Documentation . . . 91

16.6.4 Function Documentation . . . 92

16.6.5 Variable Documentation . . . 93

17 Class Documentation **93**

17.1 boost::asio::time_traits< goby::common::GobyTime > Struct Template Reference . . . 93

17.1.1 Detailed Description . . . 94

17.2 ChatCurses Class Reference . . . 94

17.2.1 Detailed Description . . . 94

17.3 goby::acomms::ABCDriver Class Reference . . . 95

17.3.1 Detailed Description . . . 95

17.3.2 Member Function Documentation . . . 95
17.4 goby::acomms::DCCLCodec Class Reference . . . 96
17.4.1 Detailed Description . . . 98
17.4.2 Member Function Documentation . . . 99
17.5 goby::acomms::DCCLDefaultBoolCodec Class Reference . . . 104
17.5.1 Detailed Description . . . 105
17.6 goby::acomms::DCCLDefaultBytesCodec Class Reference . . . 105
17.6.1 Detailed Description . . . 106
17.7 goby::acomms::DCCLDefaultEnumCodec Class Reference . . . 106
17.7.1 Detailed Description . . . 107
17.8 goby::acomms::DCCLDefaultIdentifierCodec Class Reference . . . 107
17.8.1 Detailed Description . . . 108
17.8.2 Member Function Documentation . . . 108
17.9 goby::acomms::DCCLDefaultMessageCodec Class Reference . . . 110
17.9.1 Detailed Description . . . 110
17.10 goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType > Class Template Reference . . . 111
17.10.1 Detailed Description . . . 112
17.10.2 Member Function Documentation . . . 112
17.11 goby::acomms::DCCLDefaultStringCodec Class Reference . . . 113
17.11.1 Detailed Description . . . 113
17.12 goby::acomms::DCCLException Class Reference . . . 114
17.12.1 Detailed Description . . . 114
17.13 goby::acomms::DCCLFieldCodecBase Class Reference . . . 114
17.13.1 Detailed Description . . . 118
17.13.2 Member Function Documentation . . . 118
17.14 goby::acomms::DCCLFieldCodecManager Class Reference . . . 126
17.14.1 Detailed Description . . . 126
17.14.2 Member Function Documentation . . . 127
17.15 goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable > Class Template Reference . . . 128
17.15.1 Detailed Description . . . 128
17.16 goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type > Class Template Reference . . . 129
17.16.1 Detailed Description . . . 129
17.16.2 Member Function Documentation . . . 129
17.17 goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType > Class Template Reference . . . 130
17.17.1 Detailed Description . . . 132
17.17.2 Member Function Documentation . . . 132
17.18 goby::acomms::DCCLStaticCodec< T > Class Template Reference . . . 134
17.18.1 Detailed Description . . . 135

17.19goby::acomms::DCCLTimeCodec< TimeType > Class Template Reference 135
17.19.1 Detailed Description 136
17.20goby::acomms::DCCLTypedFieldCodec< WireType, FieldType > Class Template Reference 137
17.20.1 Detailed Description 137
17.20.2 Member Function Documentation 138
17.21goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType > Class Template Reference . 140
17.21.1 Detailed Description 141
17.22goby::acomms::DCCLTypeHelper Class Reference 141
17.22.1 Detailed Description 141
17.23goby::acomms::FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE > Class Template Reference 142
17.23.1 Detailed Description 143
17.24goby::acomms::FromProtoCppTypeBase Class Reference 143
17.24.1 Detailed Description 144
17.24.2 Member Function Documentation 144
17.25goby::acomms::FromProtoCustomMessage< CustomMessage > Class Template Reference . . . 145
17.25.1 Detailed Description 146
17.26goby::acomms::FromProtoTypeBase Class Reference 146
17.26.1 Detailed Description 146
17.27goby::acomms::MACManager Class Reference 146
17.27.1 Detailed Description 147
17.27.2 Member Function Documentation 148
17.27.3 Member Data Documentation 148
17.28goby::acomms::MMDriver Class Reference 148
17.28.1 Detailed Description 149
17.28.2 Member Function Documentation 150
17.29goby::acomms::ModemDriverBase Class Reference 150
17.29.1 Detailed Description 152
17.29.2 Member Function Documentation 152
17.29.3 Member Data Documentation 154
17.30goby::acomms::QueueException Class Reference 155
17.30.1 Detailed Description 155
17.31goby::acomms::QueueManager Class Reference 156
17.31.1 Detailed Description 158
17.31.2 Member Function Documentation 158
17.31.3 Member Data Documentation 160
17.32goby::common::Colors Struct Reference 161
17.32.1 Detailed Description 161
17.33goby::common::ConfigException Class Reference 162
17.33.1 Detailed Description 162

17.34goby::common::FlexNCurses Class Reference 162
17.34.1 Detailed Description 163
17.35goby::common::FlexOstream Class Reference 163
17.35.1 Detailed Description 164
17.35.2 Member Function Documentation 164
17.36goby::common::FlexOStreamBuf Class Reference 164
17.36.1 Detailed Description 165
17.37goby::common::TermColor Class Reference 165
17.37.1 Detailed Description 166
17.38goby::Exception Class Reference 166
17.38.1 Detailed Description 166
17.39goby::moos::BluefinCommsDriver Class Reference 167
17.39.1 Detailed Description 167
17.39.2 Member Function Documentation 167
17.40goby::moos::UFldDriver Class Reference 168
17.40.1 Detailed Description 169
17.40.2 Member Function Documentation 169
17.41goby::pb::Application Class Reference 170
17.41.1 Detailed Description 171
17.41.2 Constructor & Destructor Documentation 171
17.41.3 Member Function Documentation 171
17.42goby::transitional::DCCLMessageVal Class Reference 172
17.42.1 Detailed Description 173
17.42.2 Member Function Documentation 173
17.43goby::transitional::DCCLTransitionalCodec Class Reference 176
17.43.1 Detailed Description 176
17.43.2 Constructor & Destructor Documentation 176
17.43.3 Member Function Documentation 177
17.44goby::util::LineBasedInterface Class Reference 178
17.44.1 Detailed Description 179
17.44.2 Member Function Documentation 179
17.45goby::util::SerialClient Class Reference 179
17.45.1 Detailed Description 180
17.45.2 Constructor & Destructor Documentation 180
17.46goby::util::TCPClient Class Reference 180
17.46.1 Detailed Description 181
17.46.2 Constructor & Destructor Documentation 181
17.47goby::util::TCPServer Class Reference 181
17.47.1 Detailed Description 182
17.47.2 Constructor & Destructor Documentation 182

17.48 Group Class Reference . . . 183
17.48.1 Detailed Description . . . 183
17.49 GroupSetter Class Reference . . . 183
17.49.1 Detailed Description . . . 184

18 File Documentation **184**
18.1 goby/moos/moos_protobuf_helpers.h File Reference . . . 184
18.1.1 Detailed Description . . . 185
18.1.2 Function Documentation . . . 185

19 Example Documentation **185**
19.1 acomms/amac/amac_simple/amac_simple.cpp . . . 186
19.2 acomms/chat/chat.cpp . . . 187
19.3 acomms/dccl/dccl_simple/dccl_simple.cpp . . . 190
19.4 acomms/dccl/two_message/two_message.cpp . . . 191
19.5 acomms/modemdriver/driver_simple/driver_simple.cpp . . . 192
19.6 acomms/queue/queue_simple/queue_simple.cpp . . . 194

1 Goby Underwater Autonomy Project

The Goby Underwater Autonomy Project aims to create a unified framework for multiple scientific autonomous marine vehicle collaboration, seamlessly incorporating acoustic, ethernet, wifi, and serial communications. Presently the main thrust of the project is developing a set of robust acoustic networking libraries. The Goby libraries are licensed under the GNU Lesser General Public License v3 http://www.gnu.org/licenses/lgpl.-html and the applications are licensed under the GNU General Public License v3 http://www.gnu.-org/licenses/gpl.html.

1.1 Resources

- Home page, code, bug tracking, and answers: https://launchpad.net/goby.
- User Manual: (pdf).
- Developers' Manual: (html) (pdf).
- Wiki: http://gobysoft.com/wiki.

1.2 Developer manual

- goby-acomms: An overview of Acoustic Communications Library - tackle the extremely rate limited acoustic networking problem. This library was designed with four modules that can operate independently for a developer looking integrate a specific component (e.g. just encoding/decoding) without committing to the entire goby-acomms stack.
- goby-util: Overview of Utility Libraries - provide utility functions for tasks such as logging, scientific calculations, string parsing, and serial device i/o. Goby also relies on the boost libraries http://www.boost.-org/ for many utility tasks to fill in areas where the C++ Standard Library is insufficient or unelegant.
- goby-moos: An overview of the Goby/MOOS interoperability library - classes, applications (e.g. pAcomms-Handler and iFrontSeat), and functions for interoperating between Goby and the MOOS middleware.

1.3 Publications

- T. Schneider and H. Schmidt, The Dynamic Compact Control Language: A Compact Marshalling Scheme for Acoustic Communications. IEEE OCEANS'10 / Sydney.
- T. Schneider and H. Schmidt, Goby-Acomms: A modular acoustic networking framework for short-range marine vehicle communication. Unpublished working paper.
- T. Schneider and H. Schmidt, Goby-Acomms version 2: extensible marshalling, queuing, and link layer interfacing for acoustic telemetry. 9th IFAC Conference on Manoeuvring and Control of Marine Craft '12 / Arenzano, Italy.
- T. Schneider, Advances in Integrating Autonomy with Acoustic Communications for Intelligent Networks of Marine Robots. PhD Thesis, MIT/WHOI Joint Program.

1.4 Download and Install Goby

Please visit http://gobysoft.com/wiki/InstallingGoby for help on obtaining and installing Goby.

1.5 Building Examples

Please visit http://gobysoft.com/wiki/Examples to learn about the available code examples for Goby.

1.6 Authors

Goby is developed by the Goby Developers group (https://launchpad.net/~goby-dev). The lead developer is Toby Schneider (http://gobysoft.com)

2 goby-acomms: An overview of Acoustic Communications Library

Table of Contents for goby-acomms: An overview of Acoustic Communications Library.

- Quick Start
- Overview
  - Analogy to established networking systems
  - Acoustic Communications are slow
  - Efficiency to make messages small is good
  - Total throughput unrealistic: prioritize data
  - Component model
- dccl: Encoding and decoding (Detailed documentation)
- queue: Priority based message queuing (Detailed documentation)
- modemdriver: Modem driver (Detailed documentation)
- amac: Medium Access Control (MAC) (Detailed documentation)
- Software concepts used in goby-acomms
  - Signal / Slot model for asynchronous events
  - Google Protocol Buffers
- UML models

2.1 Quick Start

To get started using the goby-acomms libraries as quickly as possible:

1. If you haven't yet, follow instructions on installing Goby: http://gobysoft.com/wiki/-InstallingGoby.
2. Identify which components you need:
   - Encoding and decoding from C++ types to bit-packed messages: dccl.
   - Queuing of DCCL messages with priority based message selection: queue.
   - A driver for interacting with the acoustic modem firmware: modemdriver.
   - Time division multiple access (TDMA) medium access control (MAC): amac.
3. Look at the "simple" code examples that accompany each component (dccl_simple.cpp, queue_simple.cpp, driver_simple.cpp, amac_simple.cpp). Then look at the example that uses all the components together: chat.cpp. The full list of examples is given in this table.
4. Refer to the rest of the documentation as needed.

Please visit https://answers.launchpad.net/goby with any questions.

```
@section acomms_overview Overview

@subsection analogy Analogy to established networking systems
```

To start on some (hopefully) common ground, let's begin with an analogy to Open Systems Initiative (OSI) networking layers in this table. For a complete description of the OSI layers see http://www.itu.int/rec/T-RE-

C-X.200-199407-I/en.

OSI Layer	Goby-Acomms library component	API class(es)	Example(s)
Application	Not yet part of Goby		MOOS Application: pAcommsHandler
Presentation	dccl: Encoding and decoding	goby::acomms::DCCL-Codec	dccl_simple.cpp two_message.cpp chat.cpp
Session	Not used, sessions are established passively.		
Transport	queue: Priority based message queuing	goby::acomms::Queue-Manager	queue_simple.cpp chat.cpp
Network	Does not yet exist. All transmissions are considered single hop, currently. Addressing routing over multiple hops is an open and pressing research problem.		
Data Link	modemdriver: Modem driver	classes derived from goby::acomms::Modem-DriverBase; e.g. goby::acomms::MM-Driver	driver_simple.cpp chat.cpp
	amac: Medium Access Control (MAC)	goby::acomms::MAC-Manager	amac_simple.cpp chat.cpp
Physical	Not part of Goby		Modem Firmware, e.g. WHOI Micro-Modem Firmware (NMEA 0183 on RS-232) (see Interface Guide)

2.1.1 Acoustic Communications are slow

Do not take the previous analogy too literally; some things we are doing here for acoustic communications (hereafter, acomms) are unconventional from the approach of networking on electromagnetic carriers (hereafter, EM networking). The difference is a vast spread in the expected throughput of a standard internet hardware carrier and acoustic communications. For example, an optical fiber can put through greater than 10 Tbps over greater than 100 km, whereas the WHOI acoustic Micro-Modem can (at best) do 5000 bps over several km. This is a difference of thirteen orders of magnitude for the bit-rate distance product!

2.1.2 Efficiency to make messages small is good

Extremely low throughput means that essentially every efficiency in bit packing messages to the smallest size possible is desirable. The traditional approach of layering (e.g. TCP/IP) creates inefficiencies as each layer wraps the message of the higher layer with its own header. See RFC3439 section 3 ("Layering Considered Harmful") for an interesting discussion of this issue http://tools.ietf.org/html/rfc3439#page-7. Thus, the "layers" of goby-acomms are more tightly interrelated than TCP/IP, for example. Higher layers depend on lower layers to carry out functions such as error checking and do not replicate this functionality.

2.1.3 Total throughput unrealistic: prioritize data

The second major difference stemming from this bandwidth constraint is that total throughput is often an unrealistic goal. The quality of the acoustic channel varies widely from place to place, and even from hour to hour as changes in the sea affect propagation of sound. This means that it is also difficult to predict what one's throughput will be at any given time.

These two considerations manifest themselves in the goby-acomms design as a priority based queuing system for the transport layer. Messages are placed in different queues based on their priority (which is determined by the designer of the message). This means that the channel is always utilized (low priority data are sent when the channel quality is high) but important messages are not swamped by low priority data. In contrast, TCP/IP considers all packets equally. Packets made from a spam email are given the same consideration as a high priority email from the President. This is a trade-off in efficiency versus simplicity that makes sense for EM networking, but does not for acoustic communications.

2.1.4 Despite all this, simplicity is good

The "law of diminishing returns" means that at some point, if we try to optimize excessively, we will end up making the system more complex without substantial gain. Thus, goby-acomms makes some concessions for the sake of simplicity:

- Numerical message fields are bounded by powers of 10, rather than 2. Humans deal much better with decimal than binary.
- User data splitting (and subsequent stitching) is not done. This is a key component of TCP/IP, but with the number of dropped packets one can expect with acomms, at the moment this does not seem like a good idea. The user is expected to provide data that is smaller or equal to the packet size of the physical layer (e.g. 32 - 256 bytes for the WHOI Micro-Modem).

2.1.5 Component model

A relatively simple component model for the goby-acomms library showing the interface classes :

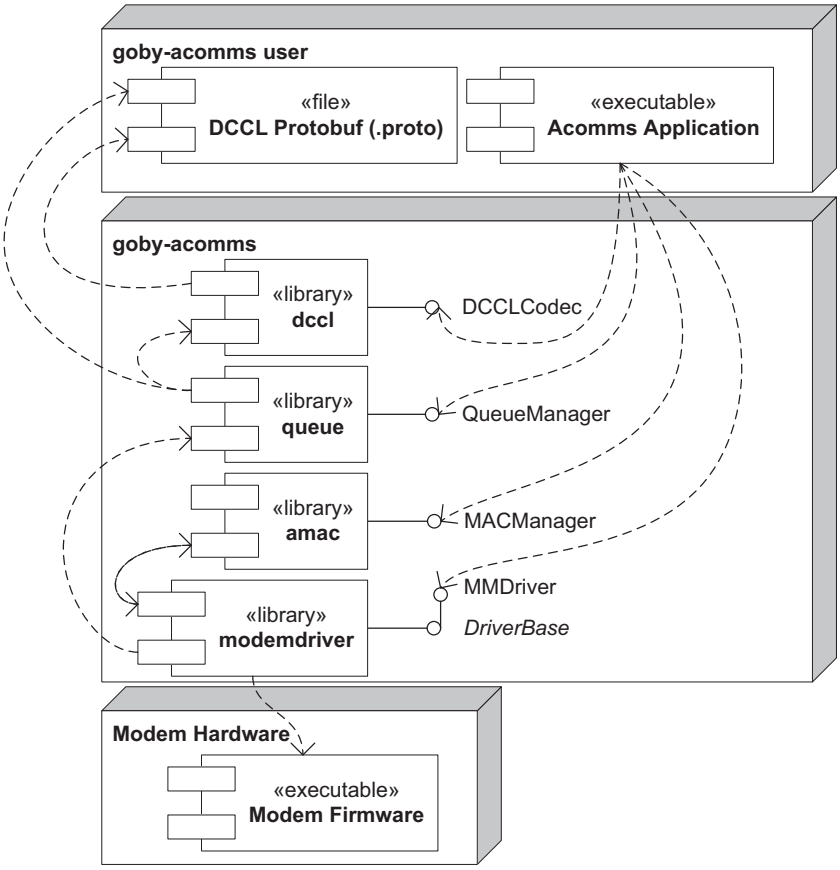


Figure 1: Basic overview of goby-acomms libraries.

For a more detailed model, see the UML models section.

2.2 dccl: Encoding and decoding

The Dynamic Compact Control Language (DCCL) provides a structure for defining messages to be sent through an acoustic modem. The messages are configured in Google Protocol Buffers and are intended to be easily reconfigurable, unlike the original CCL framework used in the REMUS vehicles and others (for information on CCL, see `http://acomms.whoi.edu/ccl/`.

Unlike the encoder / decoder provided with Google Protocol Buffers, each field (which could be a primitive type like double, int32, string or an user-defined embedded message like CTDMessage) of a DCCL message can be encoded using a DCCL built-in or user-defined encoder. This allows the codecs to be matched to the data's physical origins and thus make the most of the limited throughput available by making very small encoded messages.

Detailed documentation for goby-acomms: DCCL (Dynamic Compact Control Language).

2.3 queue: Priority based message queuing

The goby-acomms queuing (`queue`) component interacts with both the application level process and the modem driver process that talks directly to the modem.

On the application side, `queue` provides the ability for the application level process to push DCCL messages to various queues and receive messages from a remote sender that correspond to messages in the same queue (e.g. you have a queue for STATUS_MESSAGE that you can push messages to you and also receive other STATUS_MESSAGEs on). The push feature is called by the application level process and received messages are signaled to all previous bound slots (see Signal / Slot model for asynchronous events).

On the driver side, `queue` provides the modem driver with data upon request. It chooses the data to send based on dynamic priorities (and several other configuration parameters). It will also pack as many messages from the user into a single frame from the modem as possible using the DCCLCodec's repeated encoding functionality. Note, however, that `queue` will *not* split a user's data into frames (like TCP/IP). If this functionality is desired, it must be provided at the application layer. Acoustic communications are too unpredictable to reliably stitch together frames.

Detailed documentation for goby-acomms: queue (Message Priority Queuing).

2.4 modemdriver: Modem driver

The goby-acomms Modem driver component (`modemdriver`) of the Goby-Acomms library provides an interface from the rest of goby-acomms to the acoustic modem firmware. While currently the only driver publicly available is for the WHOI Micro-Modem (and for an example toy modem "ABCDriver"), this component is written in such a way that drivers for any acoustic modem that interfaces over a serial or TCP connection and can provide (or provide abstractions for) sending data directly to another modem on the link should be able to be written. Any one who is interested in writing a modem driver for another acoustic modem should get in touch with the goby project `https://launchpad.net/goby` and see Writing a new driver.

Detailed documentation for goby-acomms: modemdriver (Driver to interact with modem firmware).

2.5 amac: Medium Access Control (MAC)

The goby-acomms MAC component (amac) handles access to the shared medium, in our case the acoustic channel. We assume that we have a single (frequency) band for transmission so that if vehicles transmit simultaneously, collisions will occur between messaging. Therefore, we use time division multiple access (TDMA) schemes, or "slotting". Networks with multiple frequency bands will have to employ a different MAC scheme or augment amac for the frequency division multiple access (FDMA) scenario.

The Goby AMAC provides two basic types of TDMA:

- Decentralized: Each node initiates its own transaction at the appropriate time in the TDMA cycle. This requires reasonably well synchronized clocks (any skew must be included in the time of the slot as a guard, so skews of less than 0.1 seconds are generally acceptable.).
- Centralized (also called "polling"): For legacy support, "polling" is also provided. This is a TDMA enforced by a central computer (the "poller"). The "poller" sends a request for data from a list of nodes in sequential order. The advantage of polling is that synchronous clocks are not needed and the MAC scheme can be changed on short notice by the topside operator. Clearly this only works with modem hardware capable of third-party mediation of transmission (such as the WHOI Micro-Modem).

Detailed documentation for goby-acomms: amac (Medium Access Control).

2.6 Software concepts used in goby-acomms

2.6.1 Signal / Slot model for asynchronous events

The layers of goby-acomms use a signal / slot system for asynchronous events such as receipt of an acoustic message. Each signal can be connected (goby::acomms::connect()) to one or more slots, which are functions or member functions matching the signature of the signal. When the signal is emitted, the slots are called in order they were connected. To ensure synchronous behavior and thread-safety throughout goby-acomms, signals are only emitted during a call to a given component's API class do_work method (i.e. goby::acomms::ModemDriverBase-::do_work(), goby::acomms::QueueManager::do_work(), goby::acomms::MACManager::do_work()).

For example, if I want to receive data from `queue`, I need to connect to the signal QueueManager::signal_receive. Thus, I need to define a function or class method with the same signature:

```
void receive_data(const google::protobuf::Message& msg);
```

At startup, I then connect the signal to the slot:

```
goby::acomms::connect(&q_manager.signal_receive
      , &receive_data);
```

If instead, I was using a member function such as

```
class MyApplication
{
  public:
  void receive_data(const google::protobuf::Message& msg);

};
```

I would call connect (probably in the constructor for MyApplication) passing the pointer to the class:

```
MyApplication::MyApplication()
{
  goby::acomms::connect(&q_manager.signal_receive
      , this, &MyApplication::receive_data);
}
```

The Boost.Signals library is used without modification, so for details see http://www.boost.org/doc/libs/1_46_0/doc/html/signals.html. Member function binding is provided by Boost Bind http://www.boost.org/doc/libs/1_46_0/libs/bind/bind.html

2.6.2 Google Protocol Buffers

Google Protocol Buffers are used as a convenient way of generating data structures (basic classes with accessors, mutators). They can also be serialized efficiently, though this is not generally used within goby-acomms. Protocol buffers messages are defined in .proto files that have a C-like syntax:

```
message MyMessage
{
   optional uint32 a = 1;
   required string b = 2;
   repeated double c = 3;
}
```

The identifier "optional" means a proper MyMessage object may or may not contain that field. "required" means that a proper MyMessage always contains such a field. "repeated" means a MyMessage can contain a vector of this field (0 to n entries). The sequence number "= 1" must be unique for each field and determines the serialized format on the wire. For our purposes it is otherwise insignificant. See http://code.google.com/apis/protocolbuffers/docs/proto.html for full details.

The .proto file is pre-compiled into a C++ class that is loosely speaking (see http://code.google.com/apis/protocolbuffers/docs/reference/cpp-generated.html for precise details):

```
class MyMessage : public google::protobuf::Message
{
  public:
    MyMessage ();

    // set
    void set_a(unsigned a);
    void set_b(const std::string& b);
    void add_c(double c);

    // get
    unsigned a();
    std::string b();
    double c(int index);
    const RepeatedField<double>& c(); // RepeatedField ~= std::vector

    // has
    bool has_a();
    bool has_b();
    int c_size();
```

```
    // clear
    void clear_a();
    void clear_b();
    void clear_c();

  private:
    unsigned a_;
    std::string b_;
    RepeatedField<double> c_; // RepeatedField ~= std::vector
}
```

Clearly the .proto representation is more compact and amenable to easy modification. All the Protocol Buffers messages used in goby-acomms are placed in the goby::acomms::protobuf namespace for easy identification. This doxygen documentation does not understand Protocol Buffers language so you will need to look at the source code directly for the .proto (e.g. acomms_modem_message.proto).

2.7 UML models

Model that gives the sequence for sending a message with goby-acomms:

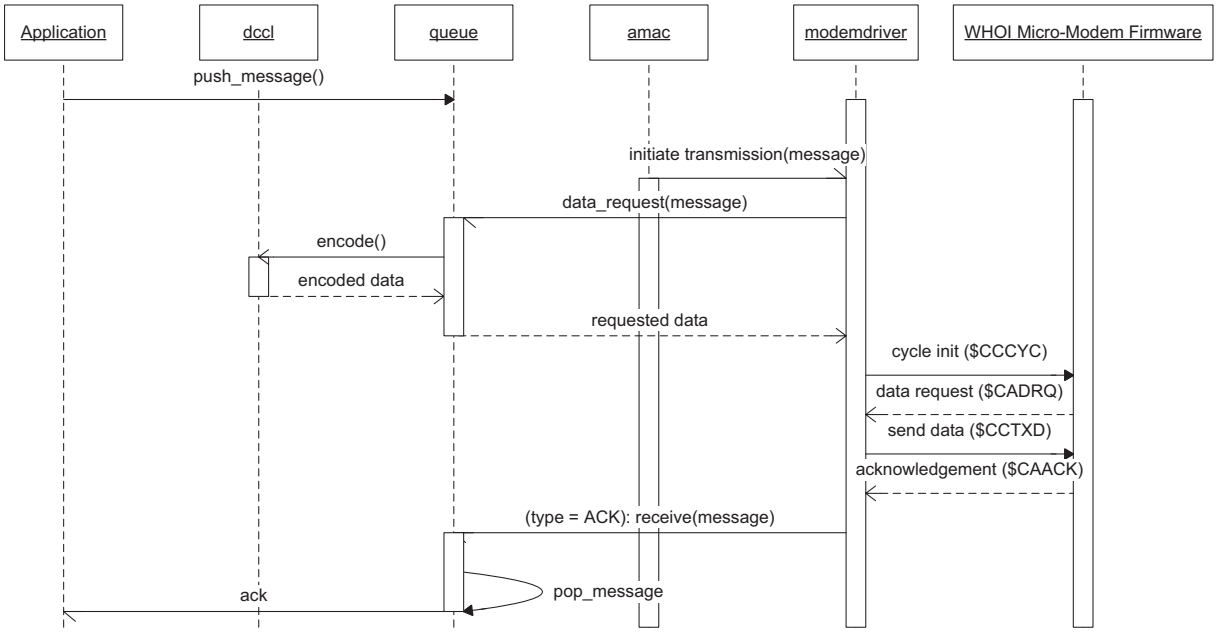


Figure 2: UML model that gives the sequence of calls required in sending a message using goby-acomms. The WHOI Micro-Modem is used as example firmware but the specific modemdriver-firmware interaction will depend on the acoustic modem used.

Model that shows the commands needed to start and keep goby-acomms running:

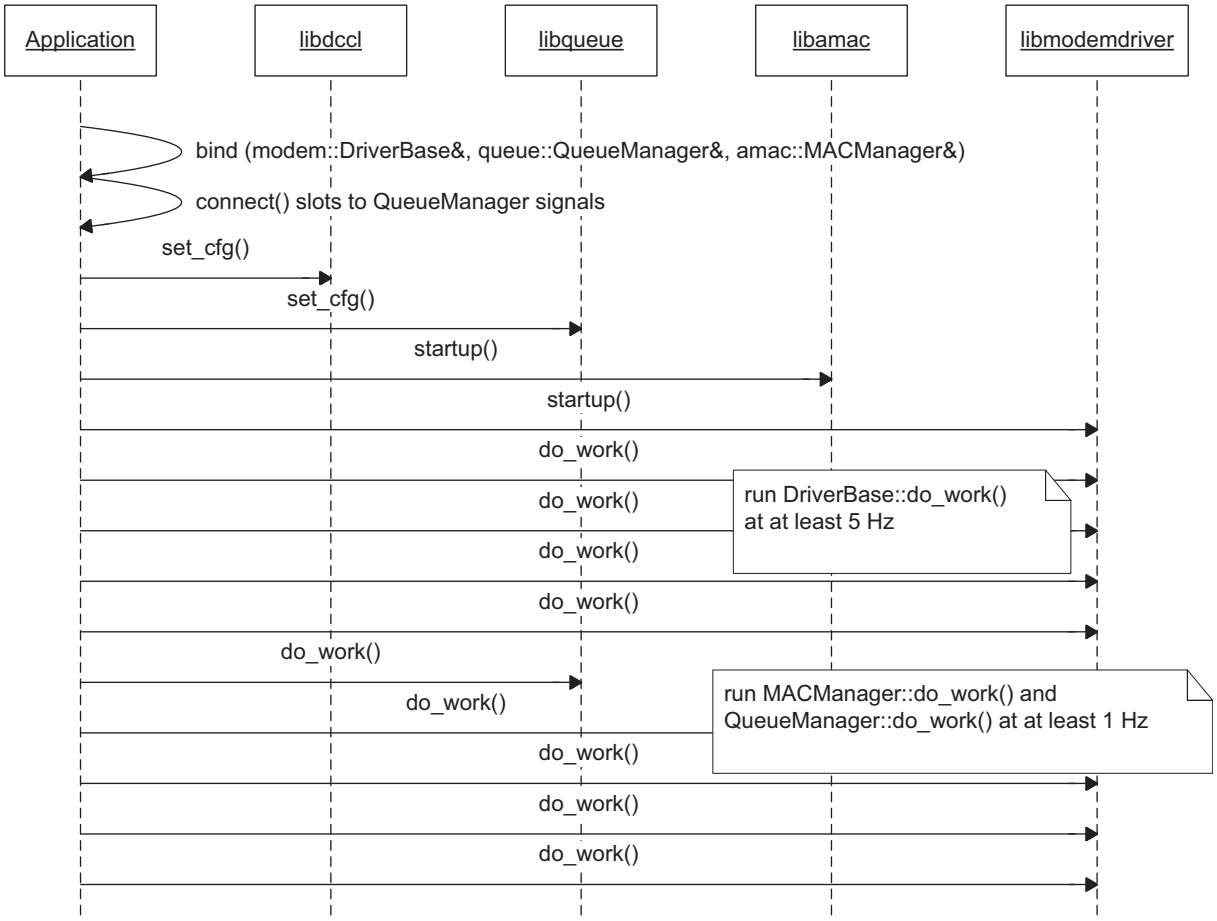


Figure 3: UML model that illustrates the set of commands needed to start up goby-acomms and keep it running.

–>

3 goby-acomms: DCCL (Dynamic Compact Control Language)

Table of contents for libdccl:

- Designing a message
- Interacting with the DCCLCodec
- DCCL Protobuf Options
- Encryption
- Example messages

Return to goby-acomms: An overview of Acoustic Communications Library.

3.1 Designing a message

DCCL uses the Google Protocol Buffers (Protobuf) language to define messages. DCCL specific components are defined as extensions to the Protobuf language message and field options. You should familiarize yourself with basic Protobuf using before reading the rest of this document: see Google Protocol Buffers and http://code.google.com/apis/protocolbuffers/docs/overview.html.

Scenario 1: Send a string command to a vehicle:

We need to send an ASCII string command to an underwater vehicle. We thus make a Protobuf message with a single string field (let's call it "telegram") to hold our command:

```
message Simple
{
  required string telegram = 1;
}
```

The "= 1" indicates that this is the first field on the wire in our DCCL message. All fields must have a unique index, but otherwise these index values are not particularly important. "required" means a valid "Simple" message always contains something for "telegram" (could be an empty string).

To turn this Protobuf message into a DCCL message, we need to add a few options. All the options are defined in acomms_option_extensions.proto so we include that:

```
import "goby/common/protobuf/option_extensions.proto";

message Simple
{
  required string telegram = 1;
}
```

At a minimum we must give a unique ID for our DCCL message and a maximum number of bytes we allow the message to be before throwing an exception when it is loaded. This allows us to ensure that we are not creating messages larger than we can send with the physical hardware. We want to have the ability to use the lowest rate WHOI Micro-Modem message size, so we pick max_bytes to be 32. We are testing so we'll use an id of 124. See http://gobysoft.org/wiki/DcclIdTable for a list of the assigned DCCL IDs.

After these additions we have:

```
import "goby/common/protobuf/option_extensions.proto";

message Simple
{
  option (dccl.msg).id = 124;
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1;
}
```

Finally, we need to pick an encoder/decoder (codec) for each field in our message. DCCL comes with defaults for all the Protobuf types. So if we don't specifically list a codec for a given field, the default is used. The default "string" codec is goby::acomms::DCCLDefaultStringCodec and is variable length. It uses one byte to list the length of the string and then up to 255 bytes to hold the contents. To ensure we stay within our bounds for the entire message (`(goby.msg).dccl.max_bytes = 32`), we have to give a maximum allowed length for a string when using the DCCLDefaultStringCodec (`(goby.field).dccl.max_length`).

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message Simple
{
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

See dccl_simple.cpp for an example of how to use this message.

Scenario 2: Send a more realistic command and receive a status message from the vehicle:

We want to be able to command our vehicle (to which we have assigned an ID number of "2") to go to a specific point on a local XY grid (meters from some known latitude / longitude), but no more than 10 kilometers from the datum. We also want to be able to turn the lights on or off, and send a short string for other new instructions. Finally, we need to be able to command a speed. Our vehicle can move no faster than 3 m/s, but its control is precise enough to handle hundredths of a m/s (wow!). It's probably easiest to make a table with our conditions:

message variable name	description	type	bounds
destination	id number of the vehicle we are commanding	int32	[0, 31]
goto_x	meters east to transit from datum	int32	[0, 10000]
goto_y	meters north to transit from datum	int32	[0, 10000]
lights_on	turn on the lights?	bool	
new_instructions	string instructions	string	no longer than 10 characters
goto_speed	transit speed (m/s)	float	[0.00, 3.00]

Taking all this into account, we form the first message (named GoToCommand) in the file two_message.proto (see Two Message example)

We choose a `dccl.id` of 125 to avoid conflicting with the message from Scenario 1 (simple.proto) and a `dccl.max_bytes` of 32 bytes to again allow sending in the WHOI Micro-Modem rate 0 packet.

Now, for the second message in two_message.proto. We want to receive the vehicle's present position and its current health, which can either be "good", "low_battery" or "abort". We make a similar table to before:

message variable name	description	type	bounds
nav_x	current vehicle position (meters east of the datum)	integer	[0, 10000]
nav_y	current vehicle position (meters north of the datum)	integer	[0, 10000]
health	vehicle state	enumeration	HEALTH_GOOD, HEALTH_LOW_BATTERY, or HEALTH_ABORT

The resulting message, can be seen under Two Message example. An example of how to use this message is given under two_message.cpp.

You can run `analyze_dccl` to view more information on your messages:

```
> analyze_dccl /path/to/two_message.proto
```

When I ran the above command I got:

```
read in: two_message.proto
=== Begin DCCLCodec ===
2 messages loaded.
= Begin GoToCommand =
Actual maximum size of message: 18 bytes / 144 bits [dccl.id head: 8, user head: 0, body: 131, padding: 5]
Allowed maximum size of message: 32 bytes / 256 bits
== Begin Header ==
== End Header ==
== Begin Body ==
GoToCommand
 required int32 destination = 1;
 :: size = 5 bit(s)
 required int32 goto_x = 3;
 :: size = 14 bit(s)
 required int32 goto_y = 4;
 :: size = 14 bit(s)
 required bool lights_on = 5;
 :: size = 1 bit(s)
 required string new_instructions = 6;
```

```
 :: min size = 8 bit(s)
 :: max size = 88 bit(s)
 required double goto_speed = 7;
 :: size = 9 bit(s)
:: min size = 51 bit(s)
:: max size = 131 bit(s)
== End Body ==
= End GoToCommand =
= Begin VehicleStatus =
Actual maximum size of message: 6 bytes / 48 bits [dccl.id head: 8, user head: 0, body: 36, padding: 4]
Allowed maximum size of message: 32 bytes / 256 bits
== Begin Header ==
== End Header ==
== Begin Body ==
VehicleStatus
 required double nav_x = 1;
 :: size = 17 bit(s)
 required double nav_y = 2;
 :: size = 17 bit(s)
 required .VehicleStatus.HealthEnum health = 3;
 :: size = 2 bit(s)
:: size = 36 bit(s)
== End Body ==
= End VehicleStatus =
=== End DCCLCodec ===
```

Besides validity checking, the most useful feature of `analyze_dccl` is the calculation of the size (in bits) of each message variable. This lets you see which fields in the message are too big. To make fields smaller, tighten up bounds.

3.2 DCCL Protobuf Options

This section gives an overview of the DCCL message and field options available for use with DCCL and the default field codecs. The full list is available in option_extensions.proto (as messages DCCLFieldOptions and DCCLMessageOptions).

DCCL message options:

name	type	default	description
id	uint32	*required*	A unique ID for each DCCL message
max_bytes	uint32	*required*	Maximum allowed size in bytes for the encoded message
codec	string	"_default"	Name of the codec to use for encoding the base message (add more codecs with goby-::acomms::DCCLFieldCodecManager::add())

DCCL field options:

name	type	default	required for codecs	description
codec	string	"_default"	optional	Name of the codec to use for encoding this field
omit	bool	false	optional	Omit this field from all DCCL encoding (has_field() will be false on receipt)

in_head	bool	false	optional	Set true for fields in the header (will *not* be encrypted, rather will be used to create the encrytion IV).
precision	int32	0	goby::acomms::DCCLDefaultNumericFieldCodec (double, float)	Number of decimal digits of precision to keep (can be negative).
min	double	0	goby::acomms::DCCLDefaultNumericFieldCodec (double, float, int32, uint32, int64, uint64, fixed32, fixed64, sfixed32, sfixed64)	Minimum value that can be encoded in this field.
max	double	0	goby::acomms::DCCLDefaultNumericFieldCodec (double, float, int32, uint32, int64, uint64, fixed32, fixed64, sfixed32, sfixed64)	Maximum value that can be encoded in this field.
static_value	string	""	goby::acomms::DCCLStaticCodec (any type)	The static value for use on decoding this placeholder field.
max_length	uint32	0	goby::acomms::DCCLDefaultStringCodec, goby::acomms::DCCLDefaultBytesCodec (string)	The maximum length of the string that can be stored in this field.
max_repeat	uint32	0	any `repeated` field	The maximum length of the `repeated` array (or vector).

3.3 Interacting with the DCCLCodec

Using the goby::acomms::DCCLCodec is a fairly straightforward endeavor (this example uses dccl_simple.cpp). First you need to get a pointer to the DCCLCodec singleton:

```
goby::acomms::DCCLCodec* codec =
      goby::acomms::DCCLCodec::get();
```

Validate all messages with the DCCLCodec to ensure all bounding constraints are met:

```
try
{
  dccl->validate<Simple>();
}
catch(DCCLException& e)
{
  std::cerr << "Oh no! " << e << std::endl;
```

```
  exit(1);
}
```

Then, to encode a message, create a Protobuf message, set its fields and pass it to goby::acomms::DCCLCodec::encode():

```
Simple message;
message.set_telegram("hello");
std::string bytes;
dccl->encode(&bytes, message);
```

`bytes` will now contain the encoded message in the form of a byte string (each char will contain a single byte of the message).

You may now send this message through whatever channel you would like.

To decode a message (stored in `bytes` as a byte string), simply pass bytes as a reference along with pointers to the Protobuf message to store the results.

```
message.Clear();
dccl->decode(bytes, &message);
```

For line by line interaction with the goby::acomms::DCCLCodec and for advanced use, investigate the code examples given in the Examples column of this table.

3.4 Encryption

Encryption of all messages can be enabled by providing a secret passphrase to the goby::acomms::protobuf::DCCLConfig object passed to goby::acomms::DCCLCodec::set_cfg(). All parties to the communication must have the same secret key.

DCCL provides AES (Rijndael) encryption for the body of the message. The header, which is sent in plain text, is hashed to form an initialization vector (IV), and the passphrase is hashed using SHA-256 to form the cipher key. You will want to make sure the header (designate fields for the header with `(goby.field).dccl.in_head = true`) is a nonce by including a constantly changing value such as time.

AES is considered secure and is used for United States top secret information.

3.5 Example messages

This section provides a listing of DCCL example Protobuf messages used in the code examples and unit tests.

3.5.1 Minimal functional DCCL message

simple.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message Simple
{
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

See Also

dccl_simple.cpp

3.5.2 Two Message example

two_message.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";
message GoToCommand
{
  option (dccl.msg).id = 125;
  option (dccl.msg).max_bytes = 32;
  required int32 destination = 1 [(dccl.field).max=31,
                                  (dccl.field).min=0,
                                  (dccl.field).precision=0];
  optional string type = 2 [(dccl.field).static_value="goto",
                            (dccl.field).codec="_static"];
  required int32 goto_x = 3 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=0];
  required int32 goto_y = 4 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=0];
  required bool lights_on = 5;
  required string new_instructions = 6 [(dccl.field).max_length=10];
  required double goto_speed = 7 [(dccl.field).max=3,
                                  (dccl.field).min=0,
                                  (dccl.field).precision=2];
}

message VehicleStatus
{
  option (dccl.msg).id = 126;
  option (dccl.msg).max_bytes = 32;
  required double nav_x = 1 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=1];
  required double nav_y = 2 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=1];
  required HealthEnum health = 3;
  enum HealthEnum {
    HEALTH_GOOD = 0;
    HEALTH_LOW_BATTERY = 1;
    HEALTH_ABORT = 2;
  }
}
```

See Also

two_message.cpp

Test1 showing all Protobuf types (using default codecs):

dccl1/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";


enum Enum1
{
  ENUM_A = 1;
  ENUM_B = 2;
  ENUM_C = 3;
}

message EmbeddedMsg1
{
  optional double val = 1 [(dccl.field).min=0,
                           (dccl.field).max=126,
                           (dccl.field).precision=3];
  optional EmbeddedMsg2 msg = 2;
}
```

```
message EmbeddedMsg2
{
  optional double val = 1 [(dccl.field).min=0,
                           (dccl.field).max=126,
                           (dccl.field).precision=2];
  optional string sval = 2 [(dccl.field).max_length=10];
  optional Enum1 enum_default = 3;
}

message TestMsg
{
  option (dccl.msg).id = 2;
  option (dccl.msg).max_bytes = 512;

  // test default enc/dec
  optional double double_default_optional = 1 [(dccl.field).min=-100,
                                       (dccl.field).max=126,
                                       (dccl.field).precision=2,
                                       (dccl.field).in_head=true];
  optional float float_default_optional = 2 [(dccl.field).min=-20,
                                       (dccl.field).max=150,
                                       (dccl.field).precision=3];
  optional int32 int32_default_optional = 3 [(dccl.field).min=-20,
                                       (dccl.field).max=3000];
  optional int64 int64_default_optional = 4 [(dccl.field).min=-710,
                                       (dccl.field).max=3000];
  optional uint32 uint32_default_optional = 5 [(dccl.field).min=0,
                                       (dccl.field).max=3000];
  optional uint64 uint64_default_optional = 6 [(dccl.field).min=5,
                                       (dccl.field).max=3000];
  optional sint32 sint32_default_optional = 7 [(dccl.field).min=-60,
                                       (dccl.field).max=3000];
  optional sint64 sint64_default_optional = 8 [(dccl.field).min=-70,
                                       (dccl.field).max=3000];
  optional fixed32 fixed32_default_optional = 9 [(dccl.field).min=0,
                                       (dccl.field).max=400];
  optional fixed64 fixed64_default_optional = 10 [(dccl.field).min=0,
                                       (dccl.field).max=3000];
  optional sfixed32 sfixed32_default_optional = 11 [(dccl.field).min=11,
                                       (dccl.field).max=3000];
  optional sfixed64 sfixed64_default_optional = 12 [(dccl.field).min=-12,
                                       (dccl.field).max=3000];

  optional bool bool_default_optional = 13;

  optional string string_default_optional = 14 [(dccl.field).max_length=8];
  optional bytes bytes_default_optional = 15 [(dccl.field).max_length=9];

  optional Enum1 enum_default_optional = 16;

  optional EmbeddedMsg1 msg_default_optional = 17;


  required double double_default_required = 21 [(dccl.field).min=-100,
                                       (dccl.field).max=126,
                                       (dccl.field).precision=2,
                                       (dccl.field).in_head=true];
  required float float_default_required = 22 [(dccl.field).min=-20,
                                       (dccl.field).max=150,
                                       (dccl.field).precision=3];
  required int32 int32_default_required = 23 [(dccl.field).min=-20,
                                       (dccl.field).max=3000];
  required int64 int64_default_required = 24 [(dccl.field).min=-710,
                                       (dccl.field).max=3000];
  required uint32 uint32_default_required = 25 [(dccl.field).min=0,
                                       (dccl.field).max=3000];
  required uint64 uint64_default_required = 26 [(dccl.field).min=5,
                                       (dccl.field).max=3000];
  required sint32 sint32_default_required = 27 [(dccl.field).min=-60,
                                       (dccl.field).max=3000];
  required sint64 sint64_default_required = 28 [(dccl.field).min=-70,
                                       (dccl.field).max=3000];
  required fixed32 fixed32_default_required = 29 [(dccl.field).min=0,
```

```
                                    (dccl.field).max=400];
  required fixed64 fixed64_default_required = 30 [(dccl.field).min=0,
                                    (dccl.field).max=3000];
  required sfixed32 sfixed32_default_required = 31 [(dccl.field).min=11,
                                    (dccl.field).max=3000];
  required sfixed64 sfixed64_default_required = 32 [(dccl.field).min=-120,
                                    (dccl.field).max=3000];

  required bool bool_default_required = 33;

  required string string_default_required = 34 [(dccl.field).max_length=8];
  required bytes bytes_default_required = 35 [(dccl.field).max_length=9];

  required Enum1 enum_default_required = 36;

  required EmbeddedMsg1 msg_default_required = 37;

  repeated double double_default_repeat = 101 [(dccl.field).min=0,
                                              (dccl.field).max=100,
                                              (dccl.field).precision=3,
                                              (dccl.field).max_repeat=4];
  repeated float float_default_repeat = 102 [(dccl.field).min=0,
                                              (dccl.field).max=100,
                                              (dccl.field).precision=3,
                                              (dccl.field).max_repeat=4];

  repeated int32 int32_default_repeat = 103 [(dccl.field).min=0,
                                            (dccl.field).max=100,
                                            (dccl.field).max_repeat=4];
  repeated int64 int64_default_repeat = 104 [(dccl.field).min=-100,
                                            (dccl.field).max=100,
                                            (dccl.field).max_repeat=4];
  repeated uint32 uint32_default_repeat = 105 [(dccl.field).min=0,
                                              (dccl.field).max=100,
                                              (dccl.field).max_repeat=4,
                                              (dccl.field).in_head=true];
  repeated uint64 uint64_default_repeat = 106 [(dccl.field).min=0,
                                              (dccl.field).max=100,
                                              (dccl.field).max_repeat=4];
  repeated sint32 sint32_default_repeat = 107 [(dccl.field).min=-60,
                                              (dccl.field).max=100,
                                              (dccl.field).max_repeat=4];
  repeated sint64 sint64_default_repeat = 108 [(dccl.field).min=-600,
                                              (dccl.field).max=100,
                                              (dccl.field).max_repeat=4];
  repeated fixed32 fixed32_default_repeat = 109 [(dccl.field).min=0,
                                                (dccl.field).max=100,
                                                (dccl.field).max_repeat=4];
  repeated fixed64 fixed64_default_repeat = 110 [(dccl.field).min=0,
                                                (dccl.field).max=100,
                                                (dccl.field).max_repeat=4];
  repeated sfixed32 sfixed32_default_repeat = 111 [(dccl.field).min=0,
                                                  (dccl.field).max=100,
                                                  (dccl.field).max_repeat=4];
  repeated sfixed64 sfixed64_default_repeat = 112 [(dccl.field).min=-500,
                                                  (dccl.field).max=100,
                                                  (dccl.field).max_repeat=4];

  repeated bool bool_default_repeat = 113 [(dccl.field).max_repeat=4];

  repeated string string_default_repeat = 114 [(dccl.field).max_length=4, (dccl.field).max_repeat=4];
  repeated bytes bytes_default_repeat = 115 [(dccl.field).max_length=4, (dccl.field).max_repeat=4];

  repeated Enum1 enum_default_repeat = 116 [(dccl.field).max_repeat=4];

  repeated EmbeddedMsg1 msg_default_repeat = 117 [(dccl.field).max_repeat=4];
}
```

See Also

dccl1/test.cpp

3.5.3 DCCL Test2 showing an embedded message encoded by a custom (non-default) codec

dccl2/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";


message CustomMsg
{
  option (dccl.msg).id = 3;
  option (dccl.msg).max_bytes = 256;
  option (dccl.msg).codec = "custom_codec";

  optional uint32 a = 1;
  optional bool b = 2;
}


message CustomMsg2
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 256;

  optional CustomMsg msg = 1;
  repeated int32 c = 3 [(dccl.field).max=100,
                        (dccl.field).min=0,
                        (dccl.field).max_repeat=4,
                        (dccl.field).codec="int32_test_codec"];
}
```

See Also

dccl2/test.cpp

3.5.4 DCCL Test3

dccl3/test.proto

```
import "goby/common/protobuf/option_extensions.proto";
import "goby/acomms/protobuf/dccl_option_extensions.proto";
import "goby/test/acomms/dccl3/header.proto";

message GobyMessage
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=10];
  required Header header = 2;
}
```

protobuf/header.proto

```
import "goby/common/protobuf/option_extensions.proto";
import "goby/acomms/protobuf/dccl_option_extensions.proto";

// required fields will be filled in for you by ApplicationBase
// if you choose not to do so yourself
message Header
{
  //
  // time
  //

  // result of goby::util::as<std::string>(goby_time())
  // e.g. "2002-01-20 23:59:59.000"
  required string time = 10 [(dccl.field).codec="_time",
```

```
                                (dccl.field).in_head=true];

  //
  // source
  //
  required string source_platform = 11 [(dccl.field).codec="_platform<->modem_id",
                                        (dccl.field).in_head=true];
  optional string source_app = 12 [(dccl.field).omit=true];

  //
  // destination
  //
  enum PublishDestination { PUBLISH_SELF = 1; PUBLISH_OTHER = 2; PUBLISH_ALL = 3; }
  optional PublishDestination dest_type = 13 [default = PUBLISH_SELF, (dccl.field).in_head=true];

  optional string dest_platform = 14 [(dccl.field).codec="_platform<->modem_id",
                                      (dccl.field).in_head=true]; // required if dest_type == other

}
```

See Also

dccl3/test.cpp

3.5.5 DCCL Test4

dccl4/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";
import "goby/test/acomms/dccl3/header.proto";

message GobyMessage1
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  optional int32 int32_val = 1 [(dccl.field).min=0, (dccl.field).max=20];
}

message GobyMessage2
{
  option (dccl.msg).id = 5;
  option (dccl.msg).max_bytes = 32;

  optional bool bool_val = 1;
}

message GobyMessage3
{
  option (dccl.msg).id = 6;
  option (dccl.msg).max_bytes = 32;

  optional string string_val = 1 [(dccl.field).max_length=10];
}
```

See Also

dccl4/test.cpp

3.5.6 DCCL Test5

dccl5/test.proto

See Also

dccl5/test.cpp

3.5.7 DCCL Test6

dccl6/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message ShortIDMsg
{
  option (dccl.msg).id = 2;
  option (dccl.msg).max_bytes = 1;
}

message ShortIDMsgWithData
{
  option (dccl.msg).id = 3;
  option (dccl.msg).max_bytes = 10;

  optional int32 in_head = 1 [(dccl.field).in_head=true, (dccl.field).min=0, (dccl.field).max=100];
  optional int32 in_body = 2 [(dccl.field).in_head=true, (dccl.field).min=0, (dccl.field).max=100];
}

message LongIDMsg
{
  option (dccl.msg).id = 10000;  option (dccl.msg).max_bytes = 2;
}

message TooLongIDMsg
{
  option (dccl.msg).id = 32768;
  option (dccl.msg).max_bytes = 32;
}

message LongIDEdgeMsg
{
  option (dccl.msg).id = 128;
  option (dccl.msg).max_bytes = 2;
}

message ShortIDEdgeMsg
{
  option (dccl.msg).id = 127;
  option (dccl.msg).max_bytes = 1;
}
```

See Also

dccl6/test.cpp

3.5.8 DCCL Test7

dccl7/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message BytesMsg
{
  option (dccl.msg).id = 10;
  option (dccl.msg).max_bytes = 32;

  required bytes req_bytes = 1 [(dccl.field).max_length=8];
  optional bytes opt_bytes = 2 [(dccl.field).max_length=8];
}
```

See Also

dccl7/test.cpp

3.5.9 DCCL Test8

dccl8/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";
import "goby/test/acomms/dccl3/header.proto";

message GobyMessage1
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  optional int32 int32_val = 1 [(dccl.field).min=0, (dccl.field).max=20];
}

message GobyMessage2
{
  option (dccl.msg).id = 5;
  option (dccl.msg).max_bytes = 32;

  optional bool bool_val = 1;
}

message GobyMessage3
{
  option (dccl.msg).id = 6;
  option (dccl.msg).max_bytes = 32;

  optional string string_val = 1 [(dccl.field).max_length=10];
}
```

See Also

dccl8/test.cpp

3.5.10 DCCL Test9

dccl9/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message MiniUser
{
  option (dccl.msg).id = 1000001;
  option (dccl.msg).max_bytes = 2;

  required uint32 user = 1 [(dccl.field).min=0,
                            (dccl.field).max=0x03FF,
                            (dccl.field).in_head=true];
}

message MiniOWTT
{
  option (dccl.msg).id = 1000002;
  option (dccl.msg).max_bytes = 2;

  required uint32 clock_mode = 1 [(dccl.field).min=0,
                                  (dccl.field).max=3,
                                  (dccl.field).in_head=true];

  required uint32 tod = 2 [(dccl.field).min=0,
                           (dccl.field).max=0x0F,
                           (dccl.field).in_head=true];

  required uint32 user = 3 [(dccl.field).min=0,
                            (dccl.field).max=0x0F,
                            (dccl.field).in_head=true];
}

message MiniAbort
```

```
{
  option (dccl.msg).id = 1000003;
  option (dccl.msg).max_bytes = 2;

  required uint32 user = 1 [(dccl.field).min=0,
                            (dccl.field).max=0x03FF,
                            (dccl.field).in_head=true];
}

message NormalDCCL
{
  option (dccl.msg).id = 1;
  option (dccl.msg).max_bytes = 32;

  required int32 a = 1 [(dccl.field).min=0,
                        (dccl.field).max=0xFFFF];
  required int32 b = 2 [(dccl.field).min=0,
                        (dccl.field).max=0xFFFF];

}
```

See Also

dccl9/test.cpp

4 goby-acomms: queue (Message Priority Queuing)

Table of Contents for `queue`:

- Understanding dynamic priority queuing
- Queuing Protobuf Options
- Interacting with the QueueManager
  - Instantiate and configure
  - Signals and (application layer) slots
  - Operation

Return to goby-acomms: An overview of Acoustic Communications Library.

4.1 Understanding dynamic priority queuing

Each queue has a base value (V_{base}) and a time-to-live (ttl) that create the priority ($P(t)$) at any given time (t):

$$P(t) = V_{base} \frac{(t - t_{last})}{ttl}$$

where t_{last} is the time of the last send from this queue.

This means for every queue, the user has control over two variables (V_{base} and ttl). V_{base} is intended to capture how important the message type is in general. Higher base values mean the message is of higher importance. The ttl governs the number of seconds the message lives from creation until it is destroyed by `queue`. The ttl also factors into the priority calculation since all things being equal (same V_{base}), it is preferable to send more time sensitive messages first. So in these two parameters, the user can capture both overall value (i.e. V_{base}) and latency tolerance (ttl) of the message queue.

The following graph illustrates the priority growth over time of three queues with different ttl and V_{base}. A message is sent every 100 seconds and the queue that is chosen is marked on the graph.

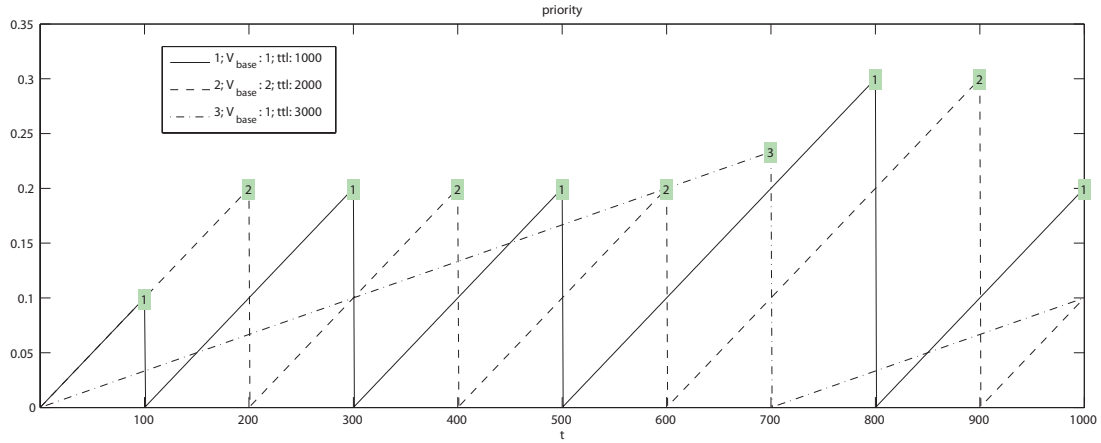


Figure 4: Graph of the growth of queueing priorities for `queue` for three different queues. A message is sent every 100 seconds from the %queue with the highest priority (numbered on the graph).

4.2 Queuing Protobuf Options

This section gives an overview of the `queue` configuration options available. The full list is available in queue.proto (as messages goby::acomms::protobuf::QueuedMessageEntry).

`Queue` message options:

name	type	default	description
ack	bool	true	Whether an acoustic acknowledgment should be requested for messages sent from this queue.
blackout_time	uint32	0	Minimum number of seconds allowed between sending messages from this queue.
max_queue	uint32	0	Allowed size of the queue before overflow. If *newest_first* is true, the oldest elements are removed upon overflow, else the newest elements are (the queue blocks). 0 is a special value signifying infinity (no maximum).

newest_first	bool	true	(true=FILO, false=FIFO) whether to send newest messages in the queue first (FILO) or not (FIFO).
ttl	int32	1800	the time in seconds a message lives after its creation before being discarded. This time-to-live also factors into the growth in priority of a queue. see value_base for the main discussion on this. 0 is a special value indicating infinite life (i.e. ttl = 0 is effectively the same as ttl = ∞)
value_base	double	1	base priority value for this message queue. priorities are calculated on a request for data by the modem (to send a message). The queue with the highest priority (and isn't in blackout) is chosen. The actual priority (P) is calculated by $P(t) = V_{base}\frac{(t-t_{last})}{ttl}$ where V_{base} is the value set here, t is the current time (in seconds), t_{last} is the time of the last send from this queue, and ttl is the ttl option. Essentially, a message with low ttl will become effective quickly again after a sent message (the priority line grows faster). See Understanding dynamic priority queuing for further discussion.

encode_on_demand	bool	false	(Advanced) enable on-demand encoding where rather than queueing data, the data request is forwarded up to the application level via the signal goby::acomms::QueueManager::signal_data_on_demand
on_demand_skew_seconds	double	1	(Advanced) if encode_on_demand == true, this sets the number of seconds before data encoded on demand are considering stale and thus must be demanded again with the signal goby::acomms::QueueManager::signal_data_on_demand. Setting this to 0 is unadvisable as it will cause many calls to goby::acomms::QueueManager::signal_data_on_demand and thus waste CPU cycles needlessly encoding.

`Queue` Role options: Queue needs to know how to address a message (the source ID and destination ID) as well as the time the message was generated. This information either read from the fields of the of the DCCL message (setting: FIELD_VALUE) or is statically configured (setting: STATIC). In the latter case, the configuration value "static_value" is set and used for every DCCL message of this type that gets queued by this QueueManager.

In the former case (the default), you can tag a given field of a DCCL message to a particular "role." This takes the place of a fixed transport layer header that protocols such as UDP use. The fields used in a given role can be anywhere within the message. The field is identified by its name (in the configuration value "field"). Submessage fields can be used by separating the field names by periods (".") until the child is a primitive type (e.g. uint32).

RoleType	allowed field types	description
SOURCE_ID	All integer types (uint32, int32, uint64, int64, ...)	The value in this field is used to represent the sending address (similar to an IP address) of the message.
DESTINATION_ID	All integer types (uint32, int32, uint64, int64, ...)	The value in this field is used to represent the destination address (similar to an IP address) of the message. 0 is reserved to indicate broadcast.
TIMESTAMP	uint64 or double	The value in this field is used as the timestamp of the message. If the type is double, it must be seconds (and fractional seconds) since the UNIX epoch (1970-01-01 midnight UTC). If it is a uint64, it must be microseconds since the UNIX epoch. This field used for expiring messages that exceed their ttl and thus must, in general, be set and correct.

4.3 Interacting with the QueueManager

4.3.1 Instantiate and configure

The goby::acomms::QueueManager is configured similarly to the goby::acomms::DCCLCodec. You need to set a unique identification number for this platform (the "modem ID") through the goby::acomms::protobuf::QueueManagerConfig .

You can configure queues by added repeated fields to the QueueManagerConfig's message_entry field, or by calling goby::acomms::QueueManager::add_queue() directly.

When using goby::acomms::QueueManager you will not likely need to use the goby::acomms::DCCLCodec directly much at all. All messages are pushed to the queues unencoded and are encoded automatically by goby::acomms::QueueManager before sending. Likewise, all messages received are decoded before being provided on the signal goby::acomms::QueueManager::signal_receive.

For example, this code configures the QueueManager with a single queue (DCCL type GobyMessage)

```
goby::acomms::protobuf::QueueManagerConfig cfg;
cfg.set_modem_id(our_id);
goby::acomms::protobuf::QueuedMessageEntry* q_entry = cfg.add_message_entry();
q_entry->set_protobuf_name("GobyMessage");
q_entry->set_newest_first(true);

goby::acomms::protobuf::QueuedMessageEntry::Role* dest_role = q_entry->add_role
      ();
dest_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::DESTINATION_ID)
      ;
dest_role->set_field("header.dest_platform");

goby::acomms::protobuf::QueuedMessageEntry::Role* time_role = q_entry->add_role
      ();
time_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::TIMESTAMP);
time_role->set_field("header.time");

goby::acomms::protobuf::QueuedMessageEntry::Role* src_role = q_entry->add_role(
      );
src_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::SOURCE_ID);
src_role->set_field("header.source_platform");
q_manager.set_cfg(cfg);
```

4.3.2 Signals and (application layer) slots

Then, you need to do a few more initialization chores:

- Connect (using goby::acomms::connect()) QueueManager signals to your application layer slots (functions or member functions that match the signal's signature). You do not need to connect a slot to a given signal if you do not need its functionality. See Signal / Slot model for asynchronous events for more on using signals and slots:
  - Received (and decoded) DCCL data: goby::acomms::QueueManager::signal_receive
  - Received acknowledgements: goby::acomms::QueueManager::signal_ack
  - Expired messages (ttl exceeded): goby::acomms::QueueManager::signal_expire
- Additional advanced features
  - Connect a slot to learn every time a queue size changes due to a new message being pushed or a message being sent: goby::acomms::QueueManager::signal_queue_size_change
  - Request that a queue be *on_demand*, that is, request data from the application layer every time the modem layer requests data (DCCL messages only). This effectively bypasses the queue and forwards the modem's data request to the application layer. Use this for sending highly time sensitive data which needs to be encoded immediately prior to sending. Set the encode_on_demand option to true for that particular Protobuf message (and if desired change the on_demand_skew_seconds). You must also connect a slot that will be executed each time data is requested to the signal goby::acomms::QueueManager::signal_data_on_demand.

4.3.3 Operation

At this point the goby::acomms::QueueManager is ready to use. At the application layer, new messages are pushed to the queues for sending using goby::acomms::QueueManager::push_message. Each queue is identified by its DCCL (Protobuf) name.

At the driver layer, messages are requested using goby::acomms::QueueManager::handle_modem_data_request and incoming messages (including acknowledgments) are published using goby::acomms::QueueManager::handle_modem_receive. If using the goby-acomms drivers (i.e. some class derived from goby::acomms::ModemDriverBase), simply call goby::acomms::bind (ModemDriverBase&, QueueManager&) and these methods (slots) will be invoked automatically from the proper driver signals.

You must run goby::acomms::QueueManager::do_work() regularly (faster than 1 Hz; 10 Hertz is good) to process expired messages (goby::acomms::QueueManager::signal_expire). All other signals are emitted in response to a driver level signal (and thus are called during a call to goby::acomms::ModemDriverBase::do_work() if using the Goby modemdriver).

See queue_simple.cpp for a basic complete example.

4.4 Example messages

This section provides a listing of `queue` example Protobuf messages used in the code examples and unit tests.

4.4.1 Minimal functional DCCL / Queue message

simple.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message Simple
{
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

4.4.2 Test1

queue1/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";


message TestMsg
{
  option (dccl.msg).id = 2;
  option (dccl.msg).max_bytes = 32;

  // test default enc/dec
  optional double double_default_optional = 1 [(dccl.field).min=-100,
                                               (dccl.field).max=126,
                                               (dccl.field).precision=2,
                                               (dccl.field).in_head=true];
  optional float float_default_optional = 2 [(dccl.field).min=-20,
                                             (dccl.field).max=150,
                                             (dccl.field).precision=3];
}
```

See Also

queue1/test.cpp

4.4.3 Test2, Test3, Test4

dccl3/test.proto

```
import "goby/common/protobuf/option_extensions.proto";
import "goby/acomms/protobuf/dccl_option_extensions.proto";
import "goby/test/acomms/dccl3/header.proto";

message GobyMessage
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=10];
  required Header header = 2;
}
```

protobuf/header.proto

```
import "goby/common/protobuf/option_extensions.proto";
import "goby/acomms/protobuf/dccl_option_extensions.proto";

// required fields will be filled in for you by ApplicationBase
// if you choose not to do so yourself
message Header
{
  //
  // time
  //

  // result of goby::util::as<std::string>(goby_time())
  // e.g. "2002-01-20 23:59:59.000"
  required string time = 10 [(dccl.field).codec="_time",
                             (dccl.field).in_head=true];

  //
  // source
  //
  required string source_platform = 11 [(dccl.field).codec="_platform<->modem_id",
                                        (dccl.field).in_head=true];
  optional string source_app = 12 [(dccl.field).omit=true];

  //
  // destination
  //
  enum PublishDestination { PUBLISH_SELF = 1; PUBLISH_OTHER = 2; PUBLISH_ALL = 3; }
  optional PublishDestination dest_type = 13 [default = PUBLISH_SELF, (dccl.field).in_head=true];

  optional string dest_platform = 14 [(dccl.field).codec="_platform<->modem_id",
                                      (dccl.field).in_head=true]; // required if dest_type == other

}
```

See Also

queue2/test.cpp
queue3/test.cpp
queue4/test.cpp

4.4.4 Test5

queue5/test.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message GobyMessage
{
  option (dccl.msg).id = 4;
  option (dccl.msg).max_bytes = 32;

  // one byte
  required int32 telegram = 1 [(dccl.field).min=0,
                               (dccl.field).max=255];
}
```

See Also

queue5/test.cpp

5 goby-acomms: amac (Medium Access Control)

Table of Contents for `amac`:

- Supported MAC schemes
- Interacting with the goby::acomms::MACManager

Return to goby-acomms: An overview of Acoustic Communications Library.

5.1 Supported MAC schemes

The Medium Access Control schemes provided by `amac` are based on Time Division Multiple Access (TDMA) where different communicators share the same bandwidth but transmit at different times to avoid conflicts. Time is divided into slots and each vehicle is given a slot to transmit on. The set of slots comprising all the vehicles is referred to here as a cycle, which repeats itself when it reaches the end. MACManager is implemented as a timer and a std::list of goby::acomms::protobuf::ModemTransmission objects. This allows you to use `amac` to create a TDMA cycle for any type of transmission (data, ping, LBL, etc.) that your modem supports.

The two variations on this scheme provided by `amac` are:

1. Decentralized: Each vehicle initiates its own transmission at the start of its slot. (goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED): Slots are set at launch and can be updated using the `std::list` insert, push, pop, erase, etc. Each vehicle can have more than one slot in the cycle. The cycles must agree across all platforms; the network designer is responsible for this. Most of the time you will want to use this mode.
2. Centralized Polling (goby::acomms::protobuf::MAC_POLLED on the master, goby::acomms::protobuf::MAC_NONE on all other nodes): The TDMA cycle is set up and operated by a centralized master modem ("poller"), which is usually the modem connected to the vehicle operator's topside. The poller initiates each transmission and thus the vehicles are not required to maintain synchronous clocks. This mode requires third-party initiation of transmissions to function.

5.2 Interacting with the goby::acomms::MACManager

To use the goby::acomms::MACManager, you need to instantiate it:

```
goby::acomms::MACManager mac;
```

Then you need to provide a callback (or "slot", not to be confused with a TDMA slot) for initiated transmissions for the signal goby::acomms::MACManager::signal_initiate_transmission. This signal will be called when the goby::acomms::MACManager determines it is time to send a message. If using `modemdriver`, simply call goby::acomms::bind(goby::acomms::MACManager&, goby::acomms::ModemDriverBase&) to bind this callback to the modem driver.

Next you need to decide which type of MAC to use: decentralized fixed or centralized polling and set the type of the goby::acomms::protobuf::MACConfig with the corresponding goby::acomms::protobuf::MACType. We also need to give goby::acomms::MACManager the vehicle's modem id (like all the other components of goby-acomms):

```
using namespace goby::acomms;
protobuf::MACConfig mac_cfg;
mac_cfg.set_type(protobuf::MAC_FIXED_DECENTRALIZED);
mac_cfg.set_modem_id(1);
```

You can also provide a set of slots in the protobuf::MACConfig to initialize the MACManager with. Otherwise, you can add them later using the std::list calls.

The usage of the goby::acomms::MACManager depends now on the type:

- goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED: All vehicles must be running goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED and share the same cycle (set of slots). Also, since each vehicle initiates its own transaction, you can use goby::acomms::QUERY_DESTINATION_ID throughout. In this example, I used the std::list push_back instead of adding the slots to the protobuf::MACConfig (see under MAC_POLLED below). Either way, you get the same result, but you can modify the std::list after startup():

```
goby::acomms::protobuf::ModemTransmission slot;
slot.set_src(1);
slot.set_dest(goby::acomms::QUERY_DESTINATION_ID
      );
slot.set_rate(0);
slot.set_type(goby::acomms::protobuf::SLOT_DATA);
slot.set_seconds(10);
mac.push_back(slot); // 1->-1@0 wait 10

slot.set_src(3);
mac.push_back(slot); // 3->-1@0 wait 10

slot.set_rate(5);
mac.push_back(slot); // 3->-1@5 wait 10

slot.set_src(4);
slot.set_rate(0);
mac.push_back(slot); // 4->-1@0 wait 10
mac.update() // important - call update() after any modifying changes to
       the MACManager underlying std::list!
```

- goby::acomms::protobuf::MAC_POLLED: On the vehicles, you do not need to run the goby::acomms::MACManager at all, or simply give it the "do nothing" goby::acomms::protobuf::MAC_NONE type. All the MAC is done on the topside (the centralized poller). On the poller, you need to manually set up a list of vehicles to be polled by adding an goby::acomms::protobuf::Slot (in the initial goby::acomms::protobuf::MACConfig object or at runtime via goby::acomms::MACManager::add_slot) for each vehicle to be polled. You can poll the same vehicle multiple times, just add more goby::acomms::protobuf::Slot objects corresponding to that vehicle. Each slot has a source, destination, rate, type (data or ping [not yet implemented]), and length (in seconds). If the source is the poller, you can set the destination to goby::acomms::QUERY_DESTINATION_ID (=-1) to let `queue` determine the next destination (based on the highest priority message to send). All goby::acomms::protobuf::Slot objects for vehicles must have a specified destination (the goby::acomms::BROADCAST_ID is a good choice or the id of the poller). For example:

```
// poll ourselves (for commands, perhaps)

goby::acomms::protobuf::ModemTransmission slot;
slot.set_src(1);
slot.set_dest(goby::acomms::QUERY_DESTINATION_ID
      );
slot.set_rate(0);
slot.set_type(goby::acomms::protobuf::ModemTransmission::DATA);
slot.SetExtension(goby::acomms::protobuf::slot_seconds, 10);
mac_cfg.add_slot(slot); // 1->-1@0 wait 10

// reuse slot
slot.set_src(3);
slot.set_dest(goby::acomms::BROADCAST_ID);
mac_cfg.add_slot(slot); // 3->0@0 wait 10

slot.set_rate(5);
mac_cfg.add_slot(slot); // 3->0@5 wait 10

slot.set_src(4);
slot.set_rate(0);
mac_cfg.add_slot(slot); // 4->0@0 wait 10
```

You can remove vehicles by a call to goby::acomms::MACManager::remove_slot or clear out the entire cycle and start over with goby::acomms::MACManager::clear_all_slots.

Then, for either MAC scheme, start the goby::acomms::MACManager running (goby::acomms::MACManager::startup with the goby::acomms::protobuf::MACConfig object), and call goby::acomms::MACManager::do_work() periodically (5 Hz is ok, 10 Hz is better).

You can modify the MAC scheme while MACManager is running. Simply use the `std::list` insert, push, pop, erase methods to changes slots (goby::acomms::protobuf::ModemTransmission objects). After any changes that invalidate std::list iterators (insert, push, pop, erase), you **must** call goby::acomms::MACManager::update() before the next call to goby::acomms::MACManager::do_work().

See amac_simple.cpp for a basic complete example.

6 goby-acomms: modemdriver (Driver to interact with modem firmware)

Table of contents for `modemdriver`:

- Abstract class: ModemDriverBase
- Protobuf Message goby::acomms::protobuf::ModemTransmission
- Writing a new driver
- WHOI Micro-Modem Driver: MMDriver

Return to goby-acomms: An overview of Acoustic Communications Library.

6.1 Abstract class: ModemDriverBase

goby::acomms::ModemDriverBase defines the core functionality for an acoustic modem. It provides

- **A serial or serial-like (over TCP) reader/writer**. This is an instantiation of an appropriate derivative of the goby::util::LineBasedInterface class which reads the physical interface (serial or TCP) to the acoustic modem. The data (assumed to be ASCII lines offset by a delimiter such as NMEA0183 or the Hayes command set [AT]) are read into a buffer for use by the goby::acomms::ModemDriverBase derived class (e.g. goby::acomms::MMDriver). The type of interface is configured using a goby::acomms::protobuf::DriverConfig. The modem is accessed by the derived class using goby::acomms::ModemDriverBase::modem_start, goby::acomms::ModemDriverBase::modem_read, goby::acomms::ModemDriverBase::modem_write, and goby::acomms::ModemDriverBase::modem_close.
- **Signals** to be called at the appropriate time by the derived class. At the application layer, either bind the modem driver to a goby::acomms::QueueManager (goby::acomms::bind(goby::acomms::ModemDriverBase&, goby::acomms::QueueManager&) or connect custom function pointers or objects to the driver layer signals.
- **Virtual functions** for starting the driver (goby::acomms::ModemDriverBase::startup), running the driver (goby::acomms::ModemDriverBase::do_work), and initiating the transmission of a message (goby::acomms::ModemDriverBase::handle_initiate_transmission). The handle_initiate_transmission slot is typically bound to goby::acomms::MACManager::signal_initiate_transmission.

6.1.1 Interacting with the goby::acomms::ModemDriverBase

To use the goby::acomms::ModemDriverBase, you need to create one of its implementations such as WHOI Micro-Modem Driver: MMDriver.

```
goby::acomms::ModemDriverBase* driver = new
      goby::acomms::MMDriver;
```

You will also need to configure the driver. At the very least this involves a serial port, baud, and modem ID (integer MAC address for the modem).

```
goby::acomms::protobuf::DriverConfig cfg;

cfg.set_serial_port("/dev/ttyS0");
cfg.set_modem_id(3);
```

Most modems will have specific other configuration that is required. For example the WHOI Micro-Modem NVRAM is set using three character strings followed by a number. This modem-specific configuration is stored as Protobuf extensions to goby::acomms::protobuf::DriverConfig, such as micromodem::protobuf::Config. If we were using the WHOI Micro-Modem and wanted to add an NVRAM configuration value we could write

```
cfg.AddExtension(micromodem::protobuf::Config::nvram_cfg, "DQF,1");
```

We need to connect any signals we are interested in. At a minimum this is goby::acomms::ModemDriverBase::signal_receive:

```
goby::acomms::connect(&driver->signal_receive
      , &handle_data_receive);
```

where handle_data_receive has the signature:

```
void handle_data_receive(const goby::acomms::protobuf::ModemTransmission&
      data_msg);
```

Next, we start up the driver with our configuration:

```
driver->startup(cfg);
```

We need to call goby::acomms::ModemDriverBase::do_work() on some reasonable frequency (greater than 5 Hz; 10 Hz is probably good). Whenever we need to transmit something, we can either directly call goby::acomms::ModemDriverBase::handle_initiate_transmission or connect goby::acomms::MACManager to do so for us on some TDMA cycle.

6.2 Protobuf Message goby::acomms::protobuf::ModemTransmission

The goby::acomms::protobuf::ModemTransmission message is used for all outgoing (sending) and incoming (receiving) messages. The message itself only contains the subset of modem functionality that every modem is expected to support (point-to-point transmission of datagrams).

All other functionality is provided by `extensions` to ModemTransmission such as those in mm_driver.proto for the WHOI Micro-Modem. These extensions provide access to additional features of the WHOI Micro-Modem (such as LBL ranging, two-way pings, and comprehensive receive statistics).

By making use of the Protobuf extensions in this way, Goby can both support unique features of a given modem while at that same time remaining general and agnostic to which modem is used when the features are shared (primarily data transfer).

6.3 Writing a new driver

All of goby-acomms is designed to be agnostic of which physical modem is used. Different modems can be supported by subclassing goby::acomms::ModemDriverBase. You should check that a driver for your modem does not yet exist before attempting to create your own.

These are the requirements of the acoustic modem:

- it communicates using a line based text duplex connection using either serial or TCP (either client or server). NMEA0183 and AT (Hayes) protocols fulfill this requirement, for example.

- it is capable of sending and verifying the accuracy (using a cyclic redundancy check or similar error checking) of fixed size datagrams (note that modems capable of variable sized datagrams also fit into this category).

Optionally, it can also support

- Acoustic acknowledgment of proper message receipt.
- Ranging to another acoustic modem or LBL beacons using time of flight measurements
- User selectable bit rates

The steps to writing a new driver include:

- Fully understand the basic usage of the new acoustic modem manually using minicom or other terminal emulator. Have a copy of the modem software interface manual handy.
- Figure out what type of configuration the modem will need. For example, the WHOI Micro-Modem is configured using string values (e.g. "SNV,1"). Extend goby::acomms::protobuf::DriverConfig to accomodate these configuration options. You will need to claim a group of extension field numbers that do not overlap with any of the drivers. The WHOI Micro-Modem driver goby::acomms::MMDriver uses extension field numbers 1000-1100 (see mm_driver.proto). You can read more about extensions in the official Google Protobuf documentation here: http://code.google.com/apis/protocolbuffers/docs/proto.-html#extensions.

  For example, if I was writing a new driver for the ABC Modem that needs to be configured using a few boolean flags, I might create a new message abc_driver.proto:

```
import "goby/acomms/protobuf/driver_base.proto"; // load up message
       DriverBaseConfig

message ABCDriverConfig
{
  extend goby.acomms.protobuf.DriverConfig
  {
    optional bool enable_foo = 1201 [ default = true ];
    optional bool enable_bar = 1202 [ default = false ];
  }
}
```

  make a note in driver_base.proto claiming extension numbers 1201 and 1202 (and others you may expect to need in the future). Extension field numbers can go up to 536,870,911 so don't worry about running out.

- Subclass goby::acomms::ModemDriverBase and overload the pure virtual methods (goby::acomms::ModemDriverBase::handle_initiate_ranging is optional). Your interface should look like this:

```
namespace goby
{
    namespace acomms
    {
        class ABCDriver : public ModemDriverBase
        {
          public:
            ABCDriver();
            void startup(const protobuf::DriverConfig& cfg);
            void shutdown();
            void do_work();
            void handle_initiate_transmission(const
       protobuf::ModemTransmission& m);

          private:
            protobuf::DriverConfig driver_cfg_; // configuration given to you
       at launch
            // rest is up to you!
        };
    }
}
```

- Fill in the methods. You are responsible for emitting the goby::acomms::ModemDriverBase signals at the appropriate times. Read on and all should be clear.

- goby::acomms::ABCDriver::ABCDriver()

```
{
  // other initialization you can do before you have your
       goby::acomms::DriverConfig configuration object
}
```

- At startup() you get your configuration from the application (e.g. pAcommsHandler)

```
void goby::acomms::ABCDriver::startup(const
      protobuf::DriverConfig& cfg)
{
    driver_cfg_ = cfg;
    // check `driver_cfg_` to your satisfaction and then start the modem
       physical interface
    if(!driver_cfg_.has_serial_baud())
        driver_cfg_.set_serial_baud(DEFAULT_BAUD);

    glog.is(DEBUG1) && glog << group("modem_out") << "ABCDriver
       configuration good. Starting modem..." << std::endl;
    ModemDriverBase::modem_start(driver_cfg_);

    // set your local modem id (MAC address)
    {
        std::stringstream raw;
        raw << "CONF,MAC:" << driver_cfg_.modem_id() << "\r\n";
        signal_and_write(raw.str());
    }


    // now set our special configuration values
    {
        std::stringstream raw;
        raw << "CONF,FOO:" << driver_cfg_.GetExtension(
      ABCDriverConfig::enable_foo) << "\r\n";
        signal_and_write(raw.str());
    }
    {
        std::stringstream raw;
        raw << "CONF,BAR:" << driver_cfg_.GetExtension(
      ABCDriverConfig::enable_bar) << "\r\n";
        signal_and_write(raw.str());
    }
} // startup
```

- At shutdown() you should make yourself ready to startup() again if necessary and stop the modem:

```
void goby::acomms::ABCDriver::shutdown()
{
    // put the modem in a low power state?
    // ...
    ModemDriverBase::modem_close();
} // shutdown
```

- handle_initiate_transmission() is called when you are expected to initiate a transmission. It *may* contain data (in the ModemTransmission::frame field). If not, you are required to request data using the goby-::acomms::ModemDriverBase::signal_data_request signal. Once you have data, you are responsible for sending it. I think a bit of code will make this clearer:

```
void goby::acomms::ABCDriver::handle_initiate_transmission
      (const protobuf::ModemTransmission& orig_msg)
{
    // copy so we can modify
    protobuf::ModemTransmission msg = orig_msg;


        // rate() can be 0 (lowest), 1, 2, 3, 4, or 5 (lowest). Map these
       integers onto real bit-rates
        // in a meaningful way (on the WHOI Micro-Modem 0 ~= 80 bps, 5 ~= 5000
       bps).
    glog.is(DEBUG1) && glog <<  group("modem_out") << "We were asked to
       transmit from "
                            << msg.src() << " to " << msg.dest()
                            << " at bitrate code " << msg.rate() << std::endl;

    // let's say ABC modem uses 500 byte packet
    msg.set_max_frame_bytes(500);

    // no data given to us, let's ask for some
    if(msg.frame_size() == 0)
        ModemDriverBase::signal_data_request(&msg);

    glog.is(DEBUG1) && glog <<  group("modem_out") << "Sending these
       data now: " << msg.frame(0) << std::endl;
```

```
    // let's say we can send at three bitrates with ABC modem: map these onto
       0-5
    const unsigned BITRATE [] = { 100, 1000, 10000, 10000, 10000, 10000};

    // I'm making up a syntax for the wire protocol...
    std::stringstream raw;
    raw << "SEND,TO:" << msg.dest()
        << ",FROM:" << msg.src()
        << ",HEX:" << hex_encode(msg.frame(0))
        << ",BITRATE:" << BITRATE[msg.rate()]
        << ",ACK:TRUE"
        << "\r\n";

    // let anyone who is interested know
    signal_and_write(raw.str());
} // handle_initiate_transmission
```

– Finally, you can use do_work() to do continuous work. You can count on it being called at 5 Hz or more (in pAcommsHandler, it is called on the MOOS AppTick). Here's where you want to read the modem incoming stream.

```
void goby::acomms::ABCDriver::do_work()
{
    std::string in;
    while(modem_read(&in))
    {
        std::map<std::string, std::string> parsed;

        // breaks `in`: "RECV,TO:3,FROM:6,HEX:ABCD015910"
        //   into `parsed`: "KEY"=>"RECV", "TO"=>"3", "FROM"=>"6",
       "HEX"=>"ABCD015910"
        try
        {
            boost::trim(in); // get whitespace off from either end
            parse_in(in, &parsed);

            // let others know about the raw feed
            protobuf::ModemRaw raw;
            raw.set_raw(in);
            ModemDriverBase::signal_raw_incoming(raw);

            protobuf::ModemTransmission msg;
            msg.set_src(goby::util::as<int32>(parsed["FROM"]));
            msg.set_dest(goby::util::as<int32>(parsed["TO"]));
            msg.set_time(goby::common::goby_time<uint64>
      ());

            glog.is(DEBUG1) && glog << group("modem_in") << in <<
      std::endl;

            if(parsed["KEY"] == "RECV")
            {
                msg.set_type(protobuf::ModemTransmission::DATA);
                msg.add_frame(hex_decode(parsed["HEX"]));
                glog.is(DEBUG1) && glog << group("modem_in") << "
      received: " << msg << std::endl;
            }
            else if(parsed["KEY"] == "ACKN")
            {
                msg.set_type(protobuf::ModemTransmission::ACK);
            }

            ModemDriverBase::signal_receive(msg);

        }
        catch(std::exception& e)
        {
            glog.is(WARN) && glog << "Bad line: " << in << std::endl;
            glog.is(WARN) && glog << "Exception: " << e.what() <<
      std::endl;
        }
    }
} // do_work
```

– Add your driver header to goby/src/acomms/modem_driver.h

– Modify driver_simple.cpp to work with your new driver.

– Add your driver to the pAcommsHandler_config.proto DriverType enumeration.

– Add your driver to the pAcommsHandler.cpp driver object creation.

The full ABC Modem example driver exists in acomms/modemdriver/abc_driver.h and acomms/modemdriver/abc_driver.cpp. A simulator for the ABC Modem exists that uses TCP to mimic a very basic set of modem commands (send data and acknowledgment). To use the ABC Modem using the driver_simple example, run this set of commands (`socat` is available in most package managers or at http://www.dest-unreach.org/socat/):

```
1. run abc_modem_simulator running on same port (as TCP server)
> abc_modem_simulator 54321
2. create fake tty terminals connected to TCP as client to port 54321
> socat -d -d -v pty,raw,echo=0,link=/tmp/ttyFAKE1 TCP:localhost:54321
> socat -d -d -v pty,raw,echo=0,link=/tmp/ttyFAKE2 TCP:localhost:54321
3. start up driver_simple
> driver_simple /tmp/ttyFAKE1 1 ABCDriver
// wait a few seconds to avoid collisions
> driver_simple /tmp/ttyFAKE2 2 ABCDriver
```

Notes:

- See goby::acomms::MMDriver for an example real implementation.
- When a message is sent to goby::acomms::BROADCAST_ID (0), it should be broadcast if the modem supports such functionality. Otherwise, the driver should throw an goby::acomms::ModemDriverException indicating that it does not support broadcast allowing the user to reconfigure their MAC / addressing scheme.

6.4 WHOI Micro-Modem Driver: MMDriver

6.4.1 Supported Functionality

The goby::acomms::MMDriver extends the goby::acomms::ModemDriverBase for the WHOI Micro-Modem acoustic modem. It is tested to work with revision 0.94.0.00 of the Micro-Modem firmware, but is known to work with older firmware (at least 0.92.0.85). It is likely to work properly with newer firmware, and any problems while using newer Micro-Modem firmware should be filed as a `bug in Goby`. The following features of the WHOI Micro-Modem are implemented, which comprise the majority of the Micro-Modem functionality:

- FSK (rate 0) data transmission
- PSK (rates 1,2,3,4,5) data transmission
- Narrowband transponder LBL ping
- REMUS transponder LBL ping
- User mini-packet 13 bit data transmission
- Two way ping

See the UML diagrams for a graphical diagram of using Goby for each of these features.

6.4.2 Micro-Modem NMEA to Goby ModemTransmission mapping

Mapping between modem_message.proto and mm_driver.proto messages and NMEA fields (see http://acomms.whoi.edu/documents/uModem%20Software%20Interface%20Guide.pdf for NMEA fields of the WHOI Micro-Modem):

Modem to Control Computer ($CA / $SN):

NMEA talker	Mapping
$CACYC	If we did not send $CCCYC, buffer data for $CADRQ by augmenting the provided ModemTransmission and calling signal_data_request: goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>() goby::acomms::protobuf::ModemTransmission.src() = ADR1 goby::acomms::protobuf::ModemTransmission.dest() = ADR2 goby::acomms::protobuf::ModemTransmission.rate() = Packet Type goby::acomms::protobuf::ModemTransmission.max_frame_bytes() = 32 for Packet Type == 0, 64 for Packet Type == 2, 256 for Packet Type == 3 or 5 goby::acomms::protobuf::ModemTransmission.max_num_frames() = 1 for Packet Type == 0, 3 for Packet Type == 2, 2 for Packet Type == 3 or 8 for Packet Type == 5
$CARXD	only for the first $CARXD for a given packet (should match with the rest though): goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>() goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::ModemTransmission::DATA goby::acomms::protobuf::ModemTransmission.src() = SRC goby::acomms::protobuf::ModemTransmission.dest() = DEST goby::acomms::protobuf::ModemTransmission.ack_requested() = ACK for each $CARXD: goby::acomms::protobuf::ModemTransmission.frame(F#-1) = goby::util::hex_decode(HH...HH)

$CAMSG	Used only to detect BAD_CRC frames ($CAMSG,BAD_CRC...). (in extension micromodem::protobuf::frame_with_bad_crc) micromodem::protobuf::frame_with_bad_crc(n) = Frame with BAD CRC (assumed next frame after last good frame). n is an integer 0,1,2,... indicating the nth reported BAD_CRC frame for this packet. (not the frame number)
$CAACK	goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>() goby::acomms::protobuf::ModemTransmission.src() = SRC goby::acomms::protobuf::ModemTransmission.dest() = DEST (first CAACK for a given packet) goby::acomms-::protobuf::ModemTransmission.acked_frame(0) = Frame#-1 (Goby starts counting at frame 0, WHOI starts with frame 1) (second CAACK for a given packet) goby::acomms-::protobuf::ModemTransmission.acked_frame(1) = Frame#-1 (third CAACK for a given packet) goby::acomms-::protobuf::ModemTransmission.acked_frame(2) = Frame#-1 ...
$CAMUA	goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::ModemTransmission::MI-CROMODEM_MINI_DATA goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>() goby::acomms::protobuf::ModemTransmission.src() = SRC goby::acomms::protobuf::ModemTransmission.dest() = DEST goby::acomms::protobuf::ModemTransmission.-frame(0) = goby::util::hex_decode(HHHH)
$CAMPR	goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>() goby::acomms::protobuf::ModemTransmission.dest() = SRC (SRC and DEST flipped to be SRC and DEST of $CCMPC) goby::acomms::protobuf::ModemTransmission.src() = DEST goby::acomms::protobuf::ModemTransmission.type() = MICROMODEM_TWO_WAY_PING (in extension micromodem::protobuf::ranging_reply) micromodem::protobuf::RangingReply.one_way_-travel_time(0) = Travel Time

$CAMPA	goby::acomms::protobuf::ModemTransmission.time() = goby::common::goby_time<uint64>() goby::acomms::protobuf::ModemTransmission.src() = SRC goby::acomms::protobuf::ModemTransmission.dest() = DEST goby::acomms::protobuf::ModemTransmission.type() = MICROMODEM_TWO_WAY_PING
$SNTTA	goby::acomms::protobuf::ModemTransmission.time() = hhmmsss.ss (converted to microseconds since 1970-01-01 00:00:00 UTC) goby::acomms::protobuf::ModemTransmission.time_source() = goby::acomms::protobuf::MODEM_TIME goby::acomms::protobuf::ModemTransmission.type() = goby::acomms::protobuf::MICROMODEM_REMUS_LBL_RANGING or goby::acomms::protobuf::MICROMODEM_NARROWBAND_LBL_RANGING (depending on which LBL type was last initiated) goby::acomms::protobuf::ModemTransmission.src() = modem ID (in extension micromodem::protobuf::ranging_reply) micromodem::protobuf::RangingReply.one_way_travel_time(0) = TA micromodem::protobuf::RangingReply.one_way_travel_time(1) = TB micromodem::protobuf::RangingReply.one_way_travel_time(2) = TC micromodem::protobuf::RangingReply.one_way_travel_time(3) = TD

$CAXST	maps onto extension micromodem::protobuf::transmit_stat of type micromodem::protobuf::TransmitStatistics. The two $CAXST messages (CYC and data) for a rate 0 FH-FSK transmission are grouped and reported at once.
$CACST	maps onto extension micromodem::protobuf::receive_stat of type micromodem::protobuf::ReceiveStatistics. The two $CACST messages for a rate 0 FH-FSK transmission are grouped and reported at once. Note that this message contains the one way time of flight for synchronous ranging (used instead of $CATOA). Also sets (which will *overwrite* goby_time() set previously): goby::acomms::protobuf::ModemTransmission.time() = TOA time (converted to microseconds since 1970-01-01 00:00:00 UTC) goby::acomms::protobuf::ModemTransmission.time_source() = goby::acomms::protobuf::MODEM_TIME
$CAREV	Not translated into any of the modem_message.proto messages. Monitored to detect excessive clock skew (between Micro-Modem clock and system clock) or reboot (INIT)
$CAERR	Not translated into any of the modem_message.proto messages. Reported to goby::glog.
$CACFG	NVRAM setting stored internally.
$CACLK	Checked against system clock and if skew is unacceptable another $CCCLK will be sent.
$CADRQ	Data request is anticipated from the $CCCYC or $CACYC and buffered. Thus it is not translated into any of the Protobuf messages.

Control Computer to Modem ($CC):

$CCTXD	SRC = goby::acomms::protobuf::ModemTransmission.src() DEST = goby::acomms::protobuf::ModemTransmission.dest() A = goby::acomms::protobuf::ModemTransmission.ack_requested() HH...HH = goby::acomms::hex_encode(goby::acomms::protobuf::ModemTransmission.frame(n)), which n is an integer 0,1,2,... corresponding to the Goby frame that this $CCTXD belongs to.

$CCCYC	Augment the ModemTransmission: goby::acomms::protobuf::ModemTransmission.max_frame_bytes() = 32 for Packet Type == 0, 64 for Packet Type == 2, 256 for Packet Type == 3 or 5 goby::acomms::protobuf::ModemTransmission.max_num_frames() = 1 for Packet Type == 0, 3 for Packet Type == 2, 2 for Packet Type == 3 or 8 for Packet Type == 5 If ADR1 == modem ID and frame_size() < max_frame_size(), buffer data for later $CADRQ by passing the ModemTransmission to signal_data_request CMD = 0 (deprecated field) ADR1 = goby::acomms::protobuf::ModemTransmission.src() ADR2 = goby::acomms::protobuf::ModemTransmission.dest() Packet Type = goby::acomms::protobuf::ModemTransmission.rate() ACK = if ADR1 == modem ID then goby::acomms::protobuf::ModemTransmission.ack_requested() else 1 Nframes = goby::acomms::protobuf::ModemTransmission.max_num_frames()

$CCCLK	Not translated from any of the modem_message.proto messages. (taken from the system time using the boost::date_time library)
$CCCFG	Not translated from any of the modem_message.proto messages. (taken from values passed to the extension micromodem::protobuf::Config::nvram_cfg of goby::acomms::protobuf::DriverConfig). If the extension micromodem::protobuf::Config::reset_nvram is set to true, $CCCFG,ALL,0 will be sent before any other $CCCFG values.)
$CCCFQ	Not translated from any of the modem_message.proto messages. $CCCFQ,ALL sent at startup.
$CCMPC	protobuf::MICROMODEM_TWO_WAY_PING == goby::acomms::protobuf::ModemTransmission.type() SRC = goby::acomms::protobuf::ModemTransmission.src() DEST = goby::acomms::protobuf::ModemTransmission.dest()
$CCPDT	protobuf::MICROMODEM_REMUS_LBL_RANGING == goby::acomms::protobuf::ModemTransmission.type() micromodem::protobuf::REMUSLBLParams type used to determine the parameters of the LBL ping. The object provided with configuration (micromodem::protobuf::Config::remus_lbl) is merged with the object provided with the ModemTransmission (micromodem::protobuf::remus_lbl) with the latter taking priority on fields that a set in both objects: GRP = 1 CHANNEL = modem ID % 4 + 1 (use four consecutive modem IDs if you need multiple vehicles pinging) SF = 0 STO = 0 Timeout = micromodem::protobuf::REMUSLBL-Params::lbl_max_range() m ∗2/ 1500 m/s ∗ 1000 ms/s + micromodem::protobuf::REMUSLBLParams-::turnaround_ms() goby::acomms::protobuf::ModemRangingRequest.-enable_beacons() is a set of four bit flags where the least significant bit is AF enable, most significant bit is DF enable. Thus b1111 == 0x0F enables all beacons AF = micromodem::protobuf::REMUSLBLParams-::enable_beacons() >> 0 & 1 BF = micromodem::protobuf::REMUSLBLParams-::enable_beacons() >> 1 & 1 CF = micromodem::protobuf::REMUSLBLParams-::enable_beacons() >> 2 & 1 DF = micromodem::protobuf::REMUSLBLParams-::enable_beacons() >> 3 & 1

$CCPNT	protobuf::MICROMODEM_NARROWBAND_LBL_RANGING == goby::acomms::protobuf::ModemTransmission.type() micromodem::protobuf::NarrowBandLBLParams type used to determine the parameters of the LBL ping. The object provided with configuration (micromodem::protobuf::Config::narrowband_lbl) is merged with the object provided with the ModemTransmission (micromodem::protobuf::narrowband_lbl) with the latter taking priority on fields that a set in both objects: Ftx = micromodem::protobuf::NarrowBandLBLParams::transmit_freq() Ttx = micromodem::protobuf::NarrowBandLBLParams::transmit_ping_ms() Trx = micromodem::protobuf::NarrowBandLBLParams::receive_ping_ms() Timeout = micromodem::protobuf::NarrowBandLBLParams::lbl_max_range() m $*$2/ 1500 m/s $*$ 1000 ms/s + micromodem::protobuf::NarrowBandLBLParams::turnaround_ms() FA = micromodem::protobuf::NarrowBandLBLParams::receive_freq(0) or 0 if receive_freq_size() $<$ 1 FB = micromodem::protobuf::NarrowBandLBLParams::receive_freq(1) or 0 if receive_freq_size() $<$ 2 FC = micromodem::protobuf::NarrowBandLBLParams::receive_freq(2) or 0 if receive_freq_size() $<$ 3 FD = micromodem::protobuf::NarrowBandLBLParams::receive_freq(3) or 0 if receive_freq_size() $<$ 4 Tflag = micromodem::protobuf::NarrowBandLBLParams::transmit_flag()

$CCMUC	SRC = goby::acomms::protobuf::ModemTransmission.src() DEST = goby::acomms::protobuf::ModemTransmission.dest() HHHH = goby::acomms::hex_encode(goby::acomms-::protobuf::ModemTransmission.frame(0)) & 0x1F

6.4.3 Sequence diagrams for various Micro-Modem features using Goby

FSK (rate 0) data transmission

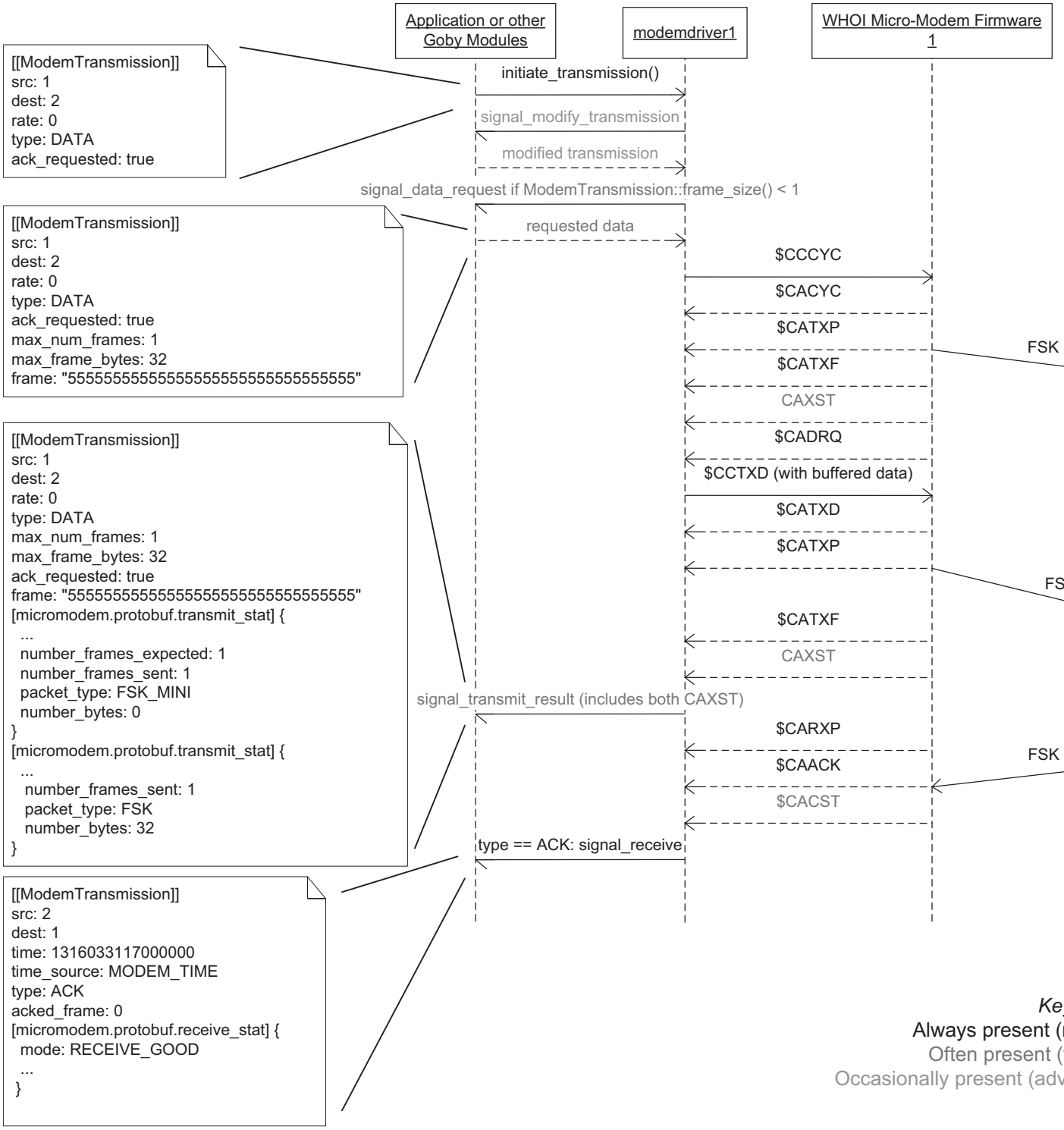


Figure 5: FSK (rate 0) data transmission

PSK (rate 2 shown, others are similar) data transmission

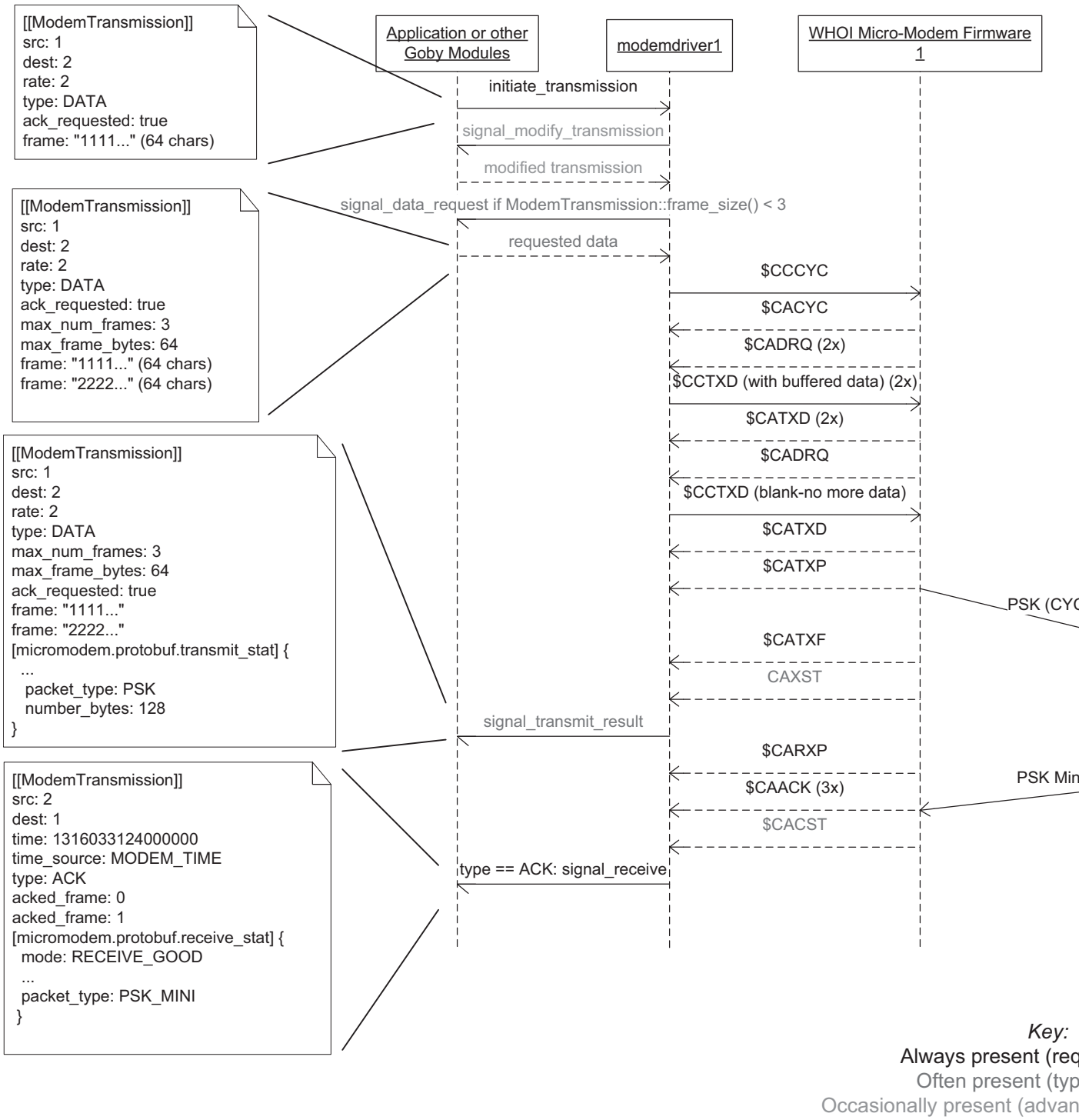


Figure 6: PSK (rate 2 shown, others are similar) data transmission

Narrowband transponder LBL ping

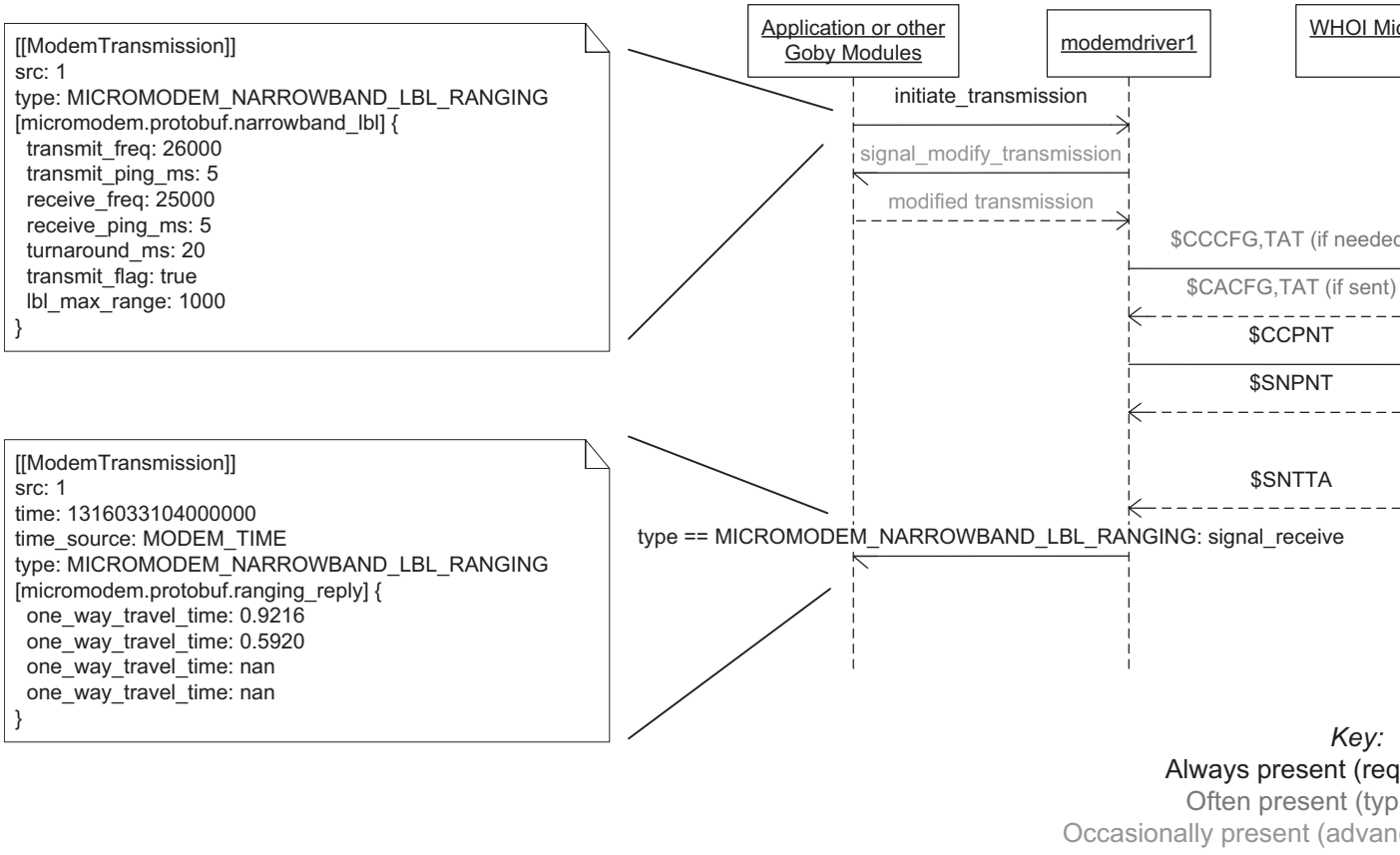


Figure 7: Narrowband transponder LBL ping

REMUS transponder LBL ping

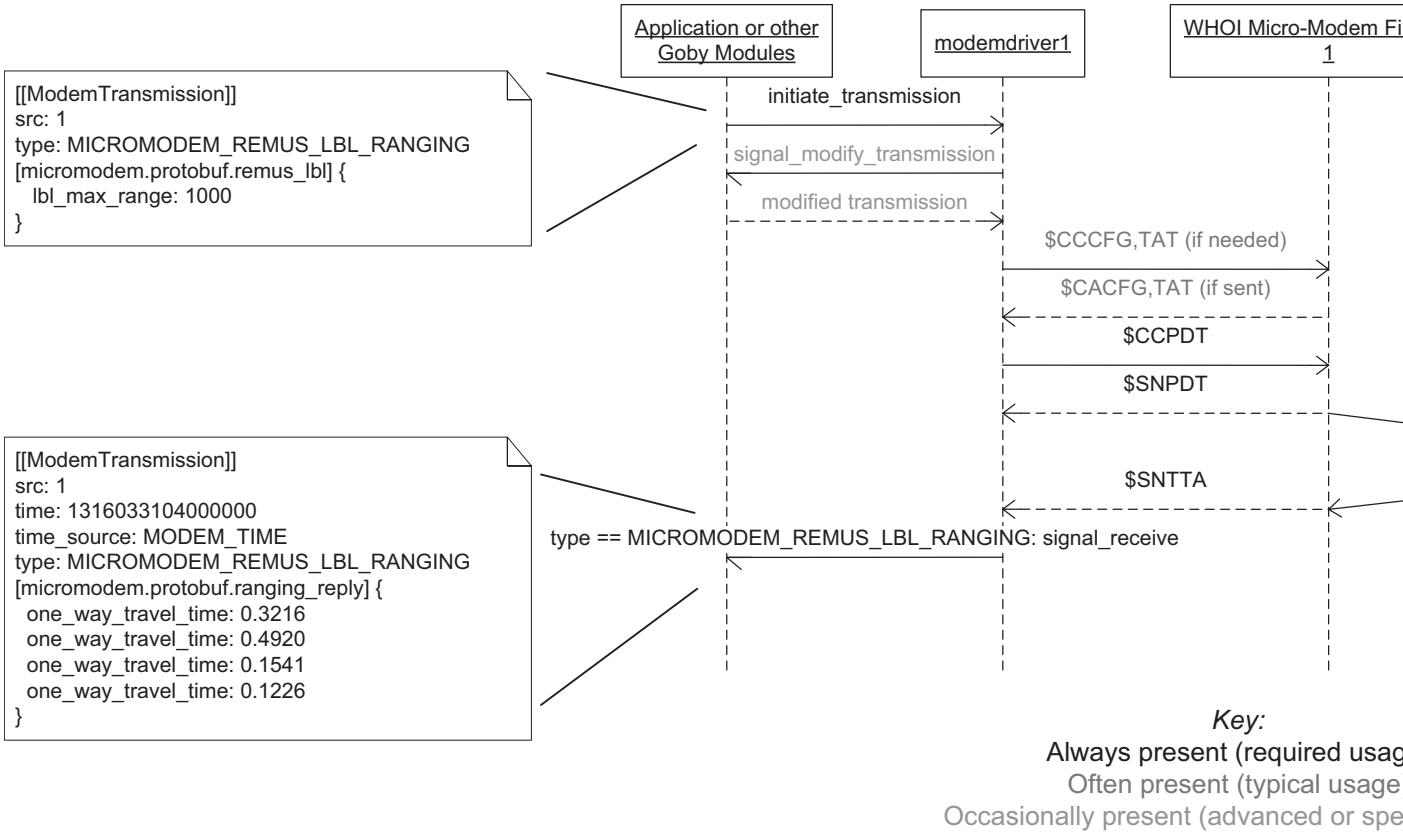


Figure 8: REMUS transponder LBL ping

User mini-packet 13 bit data transmission

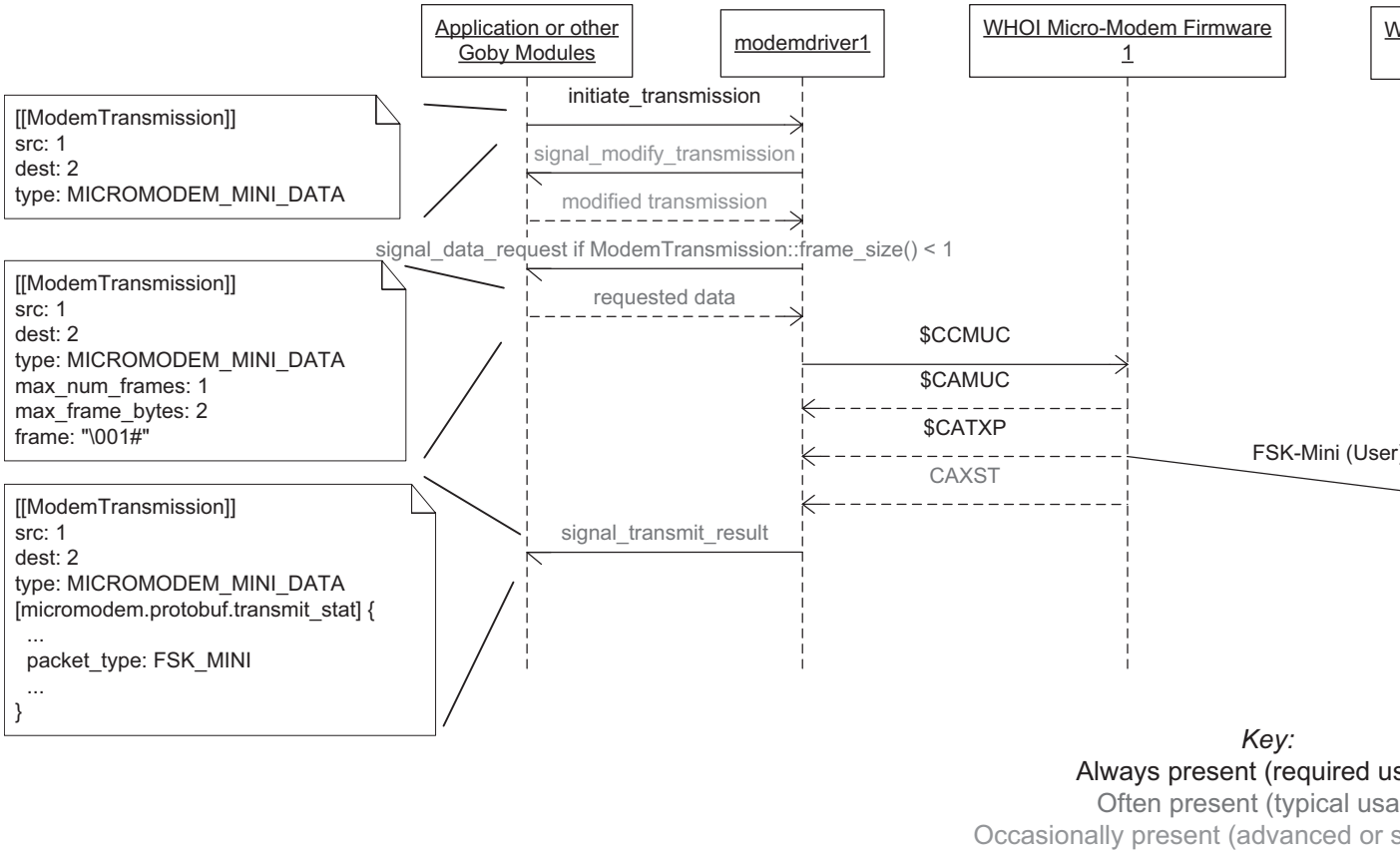


Figure 9: User mini-packet 13 bit data transmission

Two way ping

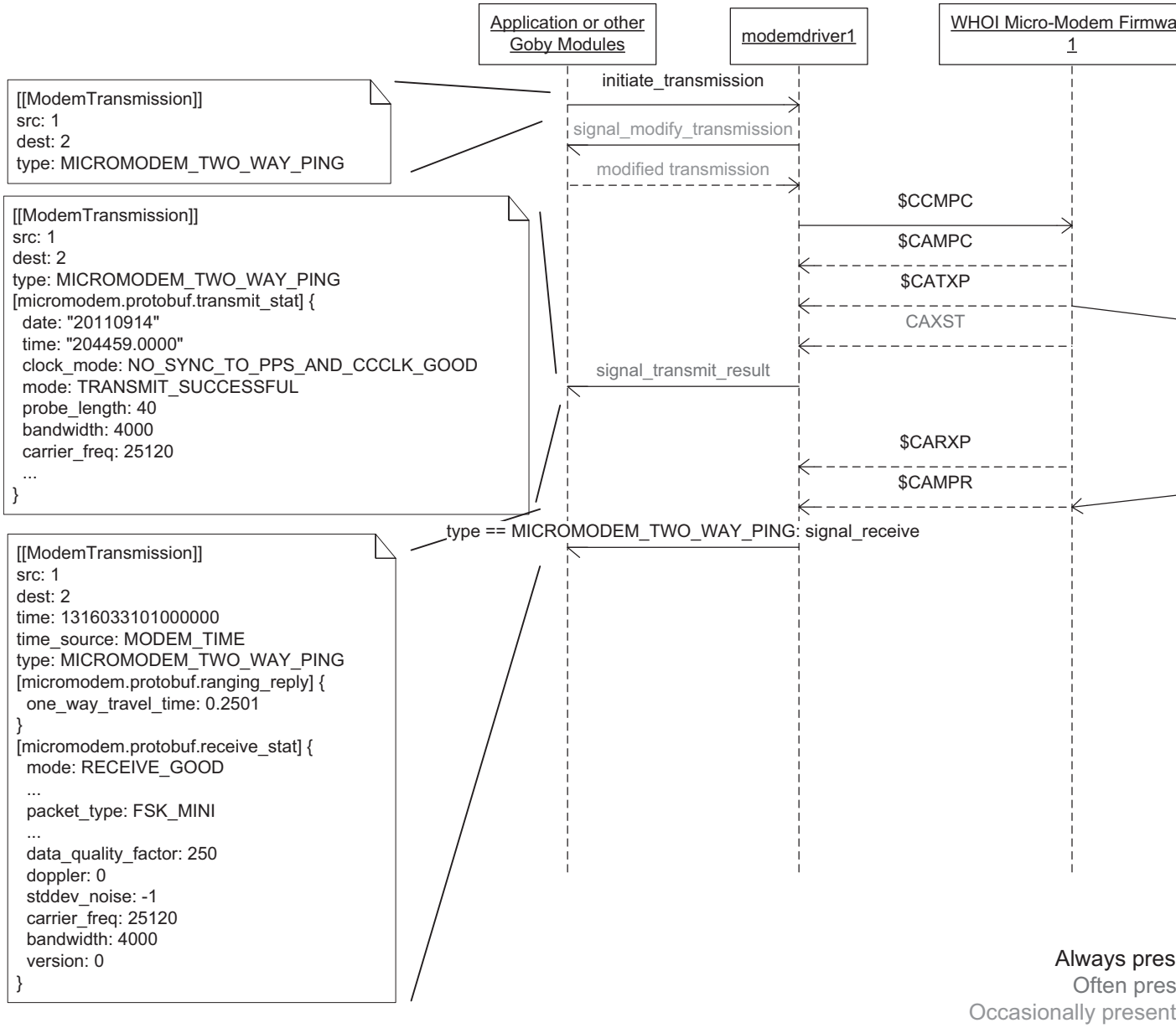


Figure 10: Two way ping

7 goby-util: Overview of Utility Libraries

Table of Contents for goby-util: Overview of Utility Libraries.

- Overview
- Logging
  - Configurable extension of std::ostream - liblogger
- TCP and Serial port communications - liblinebasedcomms


```
@section util_overview Overview

The goby-util libraries are intended to provide functions and classes for handling "utility" tasks, such as lo
```

```
@section logging Logging

Because Goby is designed first and foremost as an engineering testbed and scientific research architecture, co
The \link acomms_api goby-acomms\endlink API classes all have a constructor which can take a pointer to std::o

goby::acomms::DCCLCodec dccl(&std::cout);
```

In which case you get output (to std::cout, aka the terminal window) that looks like:

```
[ 2011-Mar-01 04:06:35.169817 ]              {dccl_enc}: cryptography enabled with given passphrase
[ 2011-Mar-01 04:06:35.170610 ]              {dccl_enc}: starting encode for TEST
[ 2011-Mar-01 04:06:35.170683 ]              {dccl_enc}:  B: bool: true
...
```

The timestamp (in Universal Coordinated Time) is given, with a group name (dccl_enc = DCCL Encoder) and finally the message. These groups are provided by using the manipulator "group". Text in the stream is a member of the given group until the next flush (std::endl or std::flush). For example:

```
// prints [ 2011-Mar-01 04:06:35.169817 ]              {my_group}: my message
std::cout << group("my_group") << "my message" << std::endl; // endl flushes my_group
```

Several other manipulators are provided:

- "debug" indicates that the buffer output is insignificant except for debugging (not useful for normal runtime)
- "warn" prints the buffer until the next flush as a warning.
- "die" is a fatal warning that calls "exit" with a non-zero code (indicating a fatal error). "die" should be used very sparingly.

```
@subsection flexostream Configurable extension of std::ostream - liblogger
goby::util::FlexOstream extends std::ostream to provide a number of extra logging features. This is generally
```

- Often it is desirable to log simultaneously to a text file (std::ofstream) and the terminal window (std::cout). goby::util::FlexOstream allows you to attach any number of streams to it, which are all written to with a single call to operator<< on the goby::util::FlexOstream object.
- Color support for ANSI terminals (std::cout and std::cerr stream objects only)
- Multiple verbosity settings for each attached stream: QUIET (display nothing to this stream), WARN (display only warnings), VERBOSE (display warnings and normal text, but not debug text), DEBUG (display warnings, normal text, and debug messages), GUI (display all messages in an NCurses terminal GUI window, splitting groups into different displays)
- Optional thread safe access using a simple lock / unlock syntax.

The best way to get used to goby:util::glogger() is to compile and play with the flexostream_simple.cpp example.

A handful of examples:

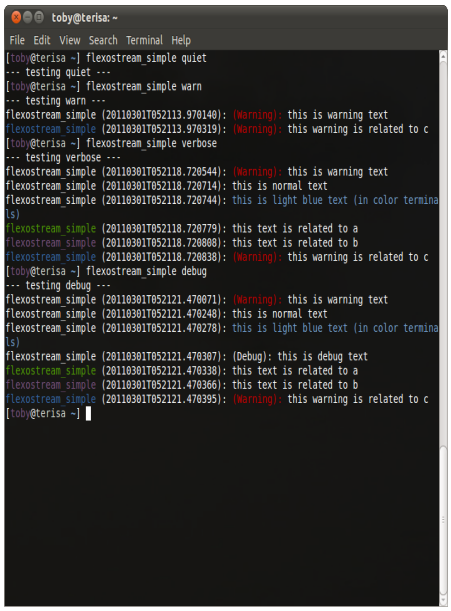

toby@terisa: ~
File Edit View Search Terminal Help
[toby@terisa ~] flexostream_simple quiet
--- testing quiet ---
[toby@terisa ~] flexostream_simple warn
--- testing warn ---
flexostream_simple (20110301T052113.970140): (Warning): this is warning text
flexostream_simple (20110301T052113.970319): (Warning): this warning is related to c
[toby@terisa ~] flexostream_simple verbose
--- testing verbose ---
flexostream_simple (20110301T052118.720544): (Warning): this is warning text
flexostream_simple (20110301T052118.720714): this is normal text
flexostream_simple (20110301T052118.720744): this is light blue text (in color termina
ls)
flexostream_simple (20110301T052118.720779): this text is related to a
flexostream_simple (20110301T052118.720808): this text is related to b
flexostream_simple (20110301T052118.720838): (Warning): this warning is related to c
[toby@terisa ~] flexostream_simple debug
--- testing debug ---
flexostream_simple (20110301T052121.470071): (Warning): this is warning text
flexostream_simple (20110301T052121.470248): this is normal text
flexostream_simple (20110301T052121.470278): this is light blue text (in color termina
ls)
flexostream_simple (20110301T052121.470307): (Debug): this is debug text
flexostream_simple (20110301T052121.470338): this text is related to a
flexostream_simple (20110301T052121.470366): this text is related to b
flexostream_simple (20110301T052121.470395): (Warning): this warning is related to c
[toby@terisa ~]

Figure 11: Example of the goby::util::glogger() output at different verbosity settings to the terminal window

Graphical user interface logger mode:

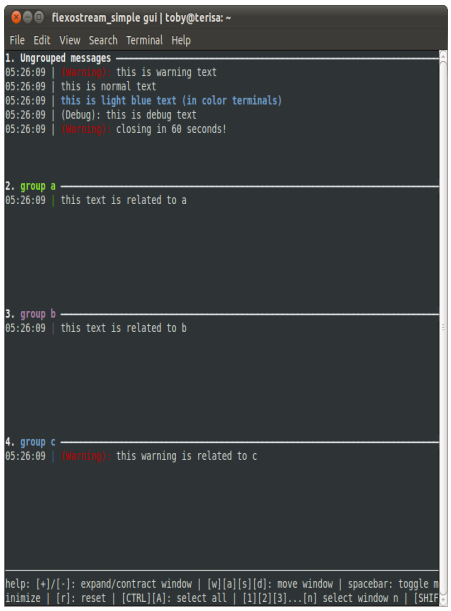

flexostream_simple gui | toby@terisa: ~
File Edit View Search Terminal Help
1. Ungrouped messages
05:26:09 | (Warning): this is warning text
05:26:09 | this is normal text
05:26:09 | this is light blue text (in color terminals)
05:26:09 | (Debug): this is debug text
05:26:09 | (Warning): closing in 60 seconds!
2. group a
05:26:09 | this text is related to a
3. group b
05:26:09 | this text is related to b
4. group c
05:26:09 | (Warning): this warning is related to c
help: [+]/[-]: expand/contract window | [w][a][s][d]: move window | spacebar: toggle m
inimize | [r]: reset | [CTRL][A]: select all | [1][2][3]...[n] select window n | [SHIF

Figure 12: Example of the goby::util::glogger() in NCurses GUI mode

Simultaneous terminal window and file logging:

```
flexostream_simple quiet|warn|verbose|debug|gui test.txt
```

test.txt:

```
[ 2011-Mar-01 05:33:26.224050 ]              {}: (Warning): this is warning text
[ 2011-Mar-01 05:33:26.224277 ]              {}: this is normal text
[ 2011-Mar-01 05:33:26.224320 ]              {}: this is light blue text (in color terminals)
[ 2011-Mar-01 05:33:26.224362 ]              {}: (Debug): this is debug text
[ 2011-Mar-01 05:33:26.224388 ]              {a}: this text is related to a
```

```
[ 2011-Mar-01 05:33:26.224429 ]              {b}: this text is related to b
[ 2011-Mar-01 05:33:26.224471 ]              {c}: (Warning): this warning is related to c

@section linebasedcomms TCP and Serial port communications - liblinebasedcomms

libutil_linebasedcomms.provides a common interface (goby::util::LineBasedInterface) for line-based (defined as
```

You should create the proper subclass for your needs:

- Serial communications: goby::util::SerialClient
- TCP Client: goby::util::TCPClient
- TCP Server: goby::util::TCPServer - all incoming messages (as read by goby::util::LineBasedInterface::readline) are interleaved in the order they are received from all connected clients. Outgoing messages are sent to all connected clients unless using goby::util::LineBasedInterface::write (const protobuf::Datagram &msg) and msg.dest() is set to a specific endpoint (ip:port, e.g. "192.168.1.101:5123").

8 goby-moos: An overview of the Goby/MOOS interoperability library

Table of Contents for goby-moos: An overview of the Goby/MOOS interoperability library.

- iFrontSeat
  - Writing a new driver for iFrontSeat
    - Overview
    - State charts
    - Example "ABC" driver

8.1 iFrontSeat

iFrontSeat is a MOOS application used to interface a Goby/MOOS community (the "backseat") running pHelmIvP with a given manufacturer's vehicle (the "frontseat"). The usage of iFrontSeat and the existing driver suite is explained in the Goby user manual (see Resources).

8.1.1 Writing a new driver for iFrontSeat

8.1.1.1 Overview

iFrontSeat is intended to interface to a wide range of vehicles using any interface (e.g. proprietary extensions of NMEA-0183). The purpose of the driver is to implement the Goby FrontSeatInterfaceBase in the language of the particular frontseat vehicle system. Minimally, these are the requirements of the frontseat:

- it can provide a navigation solution for the vehicle (minimally latitude, longitude, depth, and speed), and typically also the geo-referenced pose of the vehicle (heading, pitch, yaw).
- it can accept a desired heading, speed, and depth (at around 1 Hz) for the vehicle and carry out these commands as quickly as reasonably possible given the vehicle's dynamics.

Additionally, the frontseat may provide or consume:

- arbitrary sensor data (e.g. CTD samples, acoustic modem datagrams)
- additional special commands (e.g. buoyancy adjustment, activate/deactivate sensors, low power mode) that the backseat can command of the frontseat.

8.1.1.2 State charts

The state of iFrontSeat (as shown in the following diagram) is determined by a combination of the state of the frontseat and the state of pHelmIvP. Only the state of the frontseat must be determined by each new driver, as the state of pHelmIvP is determined by code shared by all the drivers.

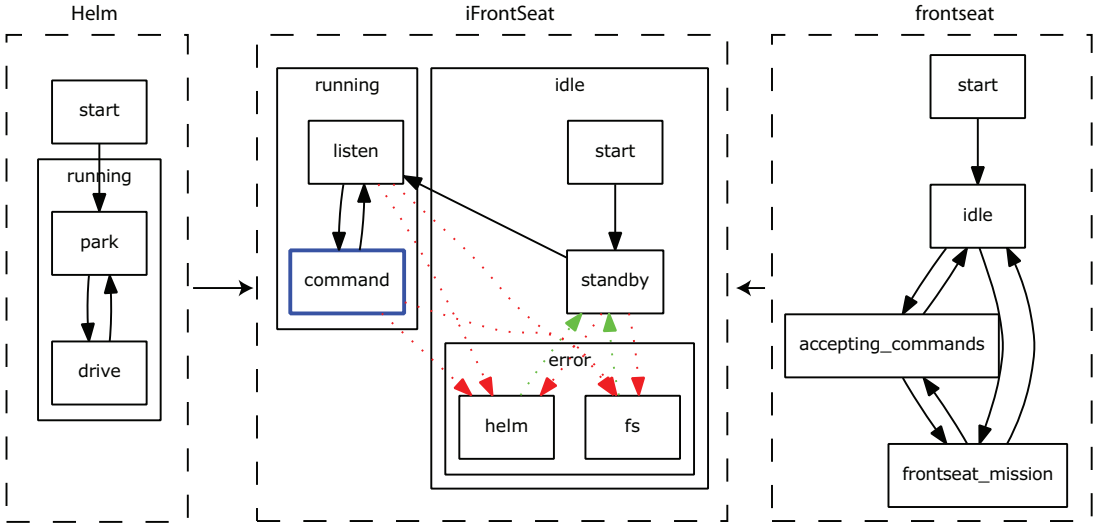


Figure 13: State charts of the iFrontSeat interface and connected ends (pHelmIvP and frontseat)

The **state of the frontseat** consists of two parallel state charts (command and data):

- Command state
  - FRONTSEAT_IDLE (required): The frontseat computer is alive and well, but is not running any mission (the vehicle is a standby mode).
  - FRONTSEAT_ACCEPTING_COMMANDS (required): The frontseat is accepting the backseat commands.
  - FRONTSEAT_NOT_CONNECTED (optional): No communication with the frontseat computer has been established (or a connection has been lost). If there is no way to tell whether the frontseat is alive at any given time, this state may not be implemented.
  - FRONTSEAT_IN_CONTROL (optional): The frontseat is running a mission and driving the vehicle but not accepting commands from the backseat. If the frontseat never runs missions without backseat control, this state may not be implemented.
- Data state (not diagrammed above)
  - frontseat_providing_data == true: The frontseat is providing all required data. What is required is determined by the specific driver, but at a minimum is the navigation solution.
  - frontseat_providing_data == false: The frontseat is not providing all required data.

The state transitions for the iFrontSeat interface states are (using the names as defined in the enumerations in moos/protobuf/frontseat.proto)

From	To	Action
Start	INTERFACE_STANDBY	Configuration Read
INTERFACE_STANDBY	INTERFACE_LISTEN	frontseat_providing_data == true
INTERFACE_LISTEN	INTERFACE_COMMAND	FRONTSEAT_ACCEPTING_COMMANDS && HELM_DRIVE
INTERFACE_COMMAND	INTERFACE_LISTEN	(FRONTSEAT_IN_CONTROL || FRONTSEAT_IDLE) && HELM_DRIVE
INTERFACE_COMMAND	INTERFACE_HELM_ERROR	HELM_NOT_RUNNING || HELM_PARK
INTERFACE_LISTEN || INTERFACE_COMMAND	INTERFACE_HELM_ERROR	HELM_PARK || if (helm_enabled) HELM_NOT_RUNNING (after timeout)
INTERFACE_LISTEN || INTERFACE_COMMAND	INTERFACE_FS_ERROR	FRONTSEAT_NOT_CONNECTED || frontseat_providing_data == false
INTERFACE_STANDBY	INTERFACE_FS_ERROR	FRONTSEAT_NOT_CONNECTED (after timeout)
INTERFACE_HELM_ERROR	INTERFACE_STANDBY	HELM_DRIVE
INTERFACE_FRONTSEAT_ERROR	INTERFACE_STANDBY	(if(ERROR_FRONTSEAT_NOT_CONNECTED) !FRONTSEAT_NOT_CONNECTED) || (if(ERROR_FRONTSEAT_NOT_PROVIDING_DATA) frontseat_providing_data == true)

8.1.1.3 Example "ABC" driver

We will show you how to a write a new driver by example. To do so, we have created a simple frontseat simulator ("abc_frontseat_simulator") that is intended to represent the real vehicle frontseat control system. The full source code for this example is given at:

- examples/moos/abc_frontseat_driver/abc_frontseat_driver.h
- examples/moos/abc_frontseat_driver/abc_frontseat_driver.cpp
- examples/moos/abc_frontseat_driver/abc_frontseat_driver_config.proto

  A complete production driver is provided by BluefinFrontSeat for the Bluefin Robotics AUVs that conform to the Bluefin Standard Payload Interface version 1.8 and newer.

  The transport for the ABC frontseat is TCP: the simulator (frontseat) listens on a given port and the driver connects to that machine and port. The wire protocol is a simple ascii line-based protocol where lines are terminated by carriage-return and newline (<CR><NL> or "\r\n"). Each message has a name (key), followed by a number of comma-delimited, colon-separated fields:

Key	Description	Direction (relative to frontseat)	Format	Example
START	Simulator initialization message	Receive	START,LAT-:{latitude decimal degrees},LON-:{longitude decimal degrees},DURAT-ION:{simulation duration seconds}	START,LAT:42.-1234,LON:-72,D-URATION:600
CTRL	Frontseat state message	Transmit	CTRL,STATE:{P-AYLOAD (if backseat control) or IDLE}	CTRL,STATE:PA-YLOAD
NAV	Navigation message generated from very primitive dynamics model (depth & heading changes are instantaneous)	Transmit	NAV,LAT:{latitude decimal degrees},LON-:{longitude decimal degrees},DEPTH-:{depth in meters},HEADIN-G:{heading in degrees},SPEED-:{speed in m/s}	NAV,LAT:42.-1234,LON:-72.-5435,DEPTH-:200,HEADING-:223,SPEED:1.4
CMD	Desired course command from backseat	Receive	CMD,HEADING-:{desired heading in degrees},SPE-ED:{desired speed in m/s},DE-PTH:{desired depth in m}	CMD,HEADING-:260,SPEED:1.-5,DEPTH:100
CMD	Reponse to last CMD	Transmit	CMD,RESULT:{O-K or ERROR}	CMD,RESULT:OK

Your driver will be (at a minimum) a C linkage function "frontseat_driver_load" and a subclass of FrontSeat-InterfaceBase. It should be compiled into a shared library (.so on Linux).

The C function is used by iFrontSeat to load your driver:

```
extern "C"
{
    FrontSeatInterfaceBase* frontseat_driver_load(iFrontSeatConfig* cfg)
    {
        return new AbcFrontSeat(*cfg);
    }
}
```

First you should decide what configuration your driver will accept. Your configuration object is an extension to the Google Protobuf message "iFrontSeatConfig". For the ABC frontseat driver, we use the abc_frontseat_-driver_config.proto file to define the configuration:

```
import "goby/moos/protobuf/frontseat_config.proto";
import "goby/common/protobuf/option_extensions.proto";

message ABCFrontSeatConfig
{
  required string tcp_address = 1;
  optional uint32 tcp_port = 2 [default = 54321];
  message StartParams
  {
    required double lat = 1;
    required double lon = 2;
```

```
    required int32 duration = 3;
  }
  required StartParams start = 3;
}

extend iFrontSeatConfig
{
  optional ABCFrontSeatConfig abc_config = 1001;
}
```

In this case, we need to know what IP address and TCP port the abc_frontseat_simulator is listening on, and the starting position of the simulator.

Next, you should fill out the virtual methods of FrontSeatInterfaceBase:

- The method "frontseat_state" reports the driver's belief of the frontseat command state (see State charts).

  ```
  goby::moos::protobuf::FrontSeatState AbcFrontSeat::frontseat_state() const
  {
      return frontseat_state_;
  } // frontseat_state
  ```

  In this case, we set the value of frontseat_status_ based on the received "CTRL" messages:

  ```
      if(parsed["KEY"] == "CTRL")
      {
          if(parsed["STATE"] == "PAYLOAD")
              frontseat_state_ = gpb::FRONTSEAT_ACCEPTING_COMMANDS;
          else if(parsed["STATE"] == "AUV")
              frontseat_state_ = gpb::FRONTSEAT_IN_CONTROL;
          else
              frontseat_state_ = gpb::FRONTSEAT_IDLE;
      }
  ```

- The method "frontseat_providing_data" reports the frontseat's data state (see State charts). It must return true if the frontseat is providing data to the driver reasonably often (where reasonable is defined by the driver). Here we set the class member variable "frontseat_providing_data_" to true each time we get a "NAV" message, and then false if we have had no "NAV" messages in the last 10 seconds.

  ```
  bool AbcFrontSeat::frontseat_providing_data() const
  {
      return frontseat_providing_data_;
  } // frontseat_providing_data
  ```

- The method "send_command_to_frontseat" is called whenever iFrontSeat needs to send a command to the frontseat. This command typically contains a desired heading, speed, and depth, but could alternatively contain a special command defined via an extension to the goby::moos::protobuf::CommandRequest message.

  ```
  void AbcFrontSeat::send_command_to_frontseat(const gpb::CommandRequest& command
        )
  {
      if(command.has_desired_course())
      {
          std::stringstream cmd_ss;
          const goby::moos::protobuf::DesiredCourse& desired_course = command.
        desired_course();
          cmd_ss << "CMD,"
                 << "HEADING:" << desired_course.heading() << ","
                 << "SPEED:" << desired_course.speed() << ","
                 << "DEPTH:" << desired_course.depth();

          write(cmd_ss.str());
          last_request_ = command;
      }
      else
      {
          glog.is(VERBOSE) && glog << "Unhandled command: " << command.
        ShortDebugString() << std::endl;
      }

  } // send_command_to_frontseat
  ```

– The method "send_data_to_frontseat" is called whenever iFrontSeat needs to send data to the frontseat. These data could include sensor readings from instruments that are directly connected to the backseat, such as a CTD or acoustic modem. Our bare-bones example frontseat doesn't require any data from the backseat, so we just leave an empty implementation here.

```
void AbcFrontSeat::send_data_to_frontseat(const gpb::FrontSeatInterfaceData&
      data)
{
    // ABC driver doesn't have any data to sent to the frontseat
} // send_data_to_frontseat
```

– The method "send_raw_to_frontseat" is called whenever an external application wants to directly control the frontseat. This can be left blank (or post a warning to the glog) if there is no need (or desire) to allow for direct control of the frontseat from external applications.

– The method "loop" is called regularly (at the AppTick of iFrontSeat) and is where you can read data from the frontseat and do other regular work.

```
void AbcFrontSeat::loop()
{
    check_connection_state();
    try_receive();

    // if we haven't gotten data for a while, set this boolean so that the
    // FrontSeatInterfaceBase class knows
    if(goby_time<double>() > last_frontseat_data_time_ + allowed_skew)
        frontseat_providing_data_ = false;
} // loop
```

Now, the final task is to call the appropriate signals in FrontSeatInterfaceBase upon receipt of data and responses to commands. The signals are called just like normal functions with the corresponding signatures. These signals (except signal_raw_to_frontseat) are typically called in response to data received in the loop() method.

– signal_data_from_frontseat: Call when a navigation solution is received from the frontseat. This may have to be merged from several messages, which is why goby::moos::protobuf::NodeStatus has the *_time_lag fields. These fields can be used to indicate the offset of certain fields from the timestamp on the message. You can use the FrontSeatInterfaceBase::compute_missing to compute the loocal fix (X, Y, Z) from the global fix (latitude, longitude, depth) or vice-versa.

– signal_command_response: Call when the frontseat acknowledges a command, if the command request includes response_requested == true. Include the success or failure of the command, and an error code (with description) if applicable.

– signal_raw_from_frontseat: Call when a raw message (e.g. "CMD,RESULT:OK") is received from the frontseat. This is for logging and debug purposes.

– signal_raw_to_frontseat: Call when a raw message (e.g. "CMD,HEADING:260,SPEED:1.5,DEPTH:100") is send to the frontseat. This is for logging and debug purposes.

For testing the ABC driver to see how it functions, you will need to run

```
abc_frontseat_simulator 54321
```

where 54321 is the port for the simulator to listen on.
Then, run iFrontSeat in a MOOS community with pHelmIvP with the following configuration:

```
ProcessConfig = iFrontSeat_bluefin
{
    common {
        verbosity: DEBUG1
    }
    [abc_config] {  #  (optional)
      tcp_address: "localhost"  #  (required)
      tcp_port: 54321  #  (optional) (default=54321)
      start { lat: 44.0888889 lon: 9.84861111 duration: 600 }
    }
}
```

You can change the start position as desired.

9 Todo List

Class goby::acomms::DCCLFieldCodecManager

(tes) Make sanity check for newly added FieldCodecs

10 Module Index

10.1 Modules

Here is a list of all modules:

API classes for the Dynamic Compact Control Language (includes writing custom encoders). 79

API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver). 79

11 Namespace Index

11.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

goby
The global namespace for the Goby project 80

goby::acomms
Objects pertaining to acoustic communications (acomms) 82

goby::common
Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc 85

goby::common::tcolor
Contains functions for adding color to Terminal window streams 88

goby::pb
Contains objects relating to the core publish / subscribe architecture provided by Goby 89

goby::transitional
Objects pertaining to transitioning from DCCLv1 to DCCLv2 89

12 Class Index

12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

boost::asio::time_traits< goby::common::GobyTime > 93

ChatCurses 94

goby::acomms::DCCLCodec 96

goby::acomms::DCCLFieldCodecBase 114

goby::acomms::DCCLFieldCodecSelector< bool, bool > 128

goby::acomms::DCCLTypedFieldCodec< bool, bool > 137

goby::acomms::DCCLTypedFixedFieldCodec< bool > 140

goby::acomms::DCCLDefaultBoolCodec 104

goby::acomms::DCCLFieldCodecSelector< CustomMsg, CustomMsg > 128

goby::acomms::DCCLTypedFieldCodec< CustomMsg, CustomMsg > 137

goby::acomms::DCCLTypedFixedFieldCodec< CustomMsg > 140

goby::acomms::DCCLFieldCodecSelector< double, double > 128

goby::acomms::DCCLTypedFieldCodec< double, double > 137

goby::acomms::DCCLTypedFixedFieldCodec< double > 140

goby::acomms::DCCLFieldCodecSelector< float, float > 128

goby::acomms::DCCLTypedFieldCodec< float, float > 137

goby::acomms::DCCLTypedFixedFieldCodec< float > 140

goby::acomms::DCCLFieldCodecSelector< goby::int32, goby::int32 > 128

goby::acomms::DCCLTypedFieldCodec< goby::int32, goby::int32 > 137

goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 > 130

goby::acomms::DCCLFieldCodecSelector< goby::uint32, goby::uint32 > 128

goby::acomms::DCCLTypedFieldCodec< goby::uint32, goby::uint32 > 137

goby::acomms::DCCLTypedFixedFieldCodec< goby::uint32 > 140

goby::acomms::DCCLFieldCodecSelector< int32, const google::protobuf::EnumValue-Descriptor * > 128

goby::acomms::DCCLTypedFieldCodec< int32, const google::protobuf::EnumValue-Descriptor * > 137

goby::acomms::DCCLTypedFixedFieldCodec< int32, const google::protobuf::Enum-ValueDescriptor * > 140

goby::acomms::DCCLDefaultNumericFieldCodec< int32, const google::protobuf:-:EnumValueDescriptor * > 111

goby::acomms::DCCLDefaultEnumCodec 106

goby::acomms::DCCLFieldCodecSelector< int32, TimeType > 128

goby::acomms::DCCLTypedFieldCodec< int32, TimeType > 137

goby::acomms::DCCLTypedFixedFieldCodec< int32, TimeType > 140

goby::acomms::DCCLDefaultNumericFieldCodec< int32, TimeType > 111

goby::acomms::DCCLTimeCodec< TimeType > 135

goby::acomms::DCCLFieldCodecSelector< Model::value_type, const google::protobuf::EnumValueDescriptor * > 128

goby::acomms::DCCLTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * > 137

goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * > 130

goby::acomms::DCCLFieldCodecSelector< Model::value_type, FieldType > 128

goby::acomms::DCCLTypedFieldCodec< Model::value_type, FieldType > 137

goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType > 130

goby::acomms::DCCLFieldCodecSelector< protobuf::CCLMDATState::GFIPitchOil, protobuf::CCLMDATState::GFIPitchOil > 128

goby::acomms::DCCLTypedFieldCodec< protobuf::CCLMDATState::GFIPitchOil, protobuf::CCLMDATState::GFIPitchOil > 137

goby::acomms::DCCLTypedFixedFieldCodec< protobuf::CCLMDATState::GFIPitchOil > 140

goby::acomms::DCCLFieldCodecSelector< std::string, std::string > 128

goby::acomms::DCCLTypedFieldCodec< std::string > 137

goby::acomms::DCCLDefaultBytesCodec 105

goby::acomms::DCCLDefaultStringCodec 113

goby::acomms::DCCLFieldCodecSelector< T, T > 128

goby::acomms::DCCLTypedFieldCodec< T, T > 137

goby::acomms::DCCLTypedFixedFieldCodec< T > 140

goby::acomms::DCCLStaticCodec< T > 134

goby::acomms::DCCLFieldCodecSelector< uint32, uint32 > 128

goby::acomms::DCCLTypedFieldCodec< uint32 > 137

goby::acomms::DCCLDefaultIdentifierCodec 107

goby::acomms::DCCLTypedFieldCodec< uint32, uint32 > 137

goby::acomms::DCCLTypedFixedFieldCodec< uint32, uint32 > 140

goby::acomms::DCCLDefaultNumericFieldCodec< uint32 > 111

goby::acomms::DCCLFieldCodecSelector< uint64, uint64 > 128

goby::acomms::DCCLTypedFieldCodec< uint64, uint64 > 137

goby::acomms::DCCLTypedFixedFieldCodec< uint64 > 140

goby::acomms::DCCLFieldCodecSelector< WireType, FieldType > 128

goby::acomms::DCCLTypedFieldCodec< WireType, FieldType > 137

goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType > 130

goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType > 140

goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType > 111

goby::acomms::DCCLDefaultMessageCodec 110

goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable > 128

goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type > 129

goby::acomms::DCCLFieldCodecManager 126

goby::acomms::DCCLTypeHelper 141

goby::acomms::FromProtoCppTypeBase 143

goby::acomms::FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE > 142

goby::acomms::FromProtoCustomMessage< CustomMessage > 145

goby::acomms::FromProtoTypeBase 146

goby::acomms::MACManager 146

goby::acomms::ModemDriverBase 150

goby::acomms::ABCDriver 95

goby::acomms::MMDriver 148

goby::moos::BluefinCommsDriver 167

goby::moos::UFldDriver 168

goby::acomms::QueueManager 156

goby::common::Colors 161

goby::common::FlexNCurses 162

goby::common::FlexOstream 163

goby::common::FlexOStreamBuf 164

goby::common::TermColor 165

goby::Exception 166

goby::acomms::DCCLException 114

goby::acomms::QueueException 155

goby::common::ConfigException 162

goby::pb::Application 170

goby::transitional::DCCLMessageVal 172

goby::transitional::DCCLTransitionalCodec 176

goby::util::LineBasedInterface 178

goby::util::TCPServer 181

goby::util::SerialClient 179

goby::util::TCPClient 180

Group 183

GroupSetter 183

13 Class Index

13.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

boost::asio::time_traits< goby::common::GobyTime >
Time traits specialised for GobyTime 93

ChatCurses
Terminal GUI for a chat window (lower box to type and upper box to receive messages). Part of the chat.cpp example 94

goby::acomms::ABCDriver
API to the imaginary ABC modem (as an example how to write drivers) 95

goby::acomms::DCCLCodec
API to the Dynamic CCL Codec 96

goby::acomms::DCCLDefaultBoolCodec
Provides a bool encoder. Uses 1 bit if field is `required`, 2 bits if `optional` 104

goby::acomms::DCCLDefaultBytesCodec
Provides an fixed length byte string encoder 105

goby::acomms::DCCLDefaultEnumCodec
Provides an enum encoder. This converts the enumeration to an integer (based on the enumeration *index* (not its *value*) and uses DCCLDefaultNumericFieldCodec to encode the integer 106

goby::acomms::DCCLDefaultIdentifierCodec
Provides the default 1 byte or 2 byte DCCL ID codec 107

goby::acomms::DCCLDefaultMessageCodec
Provides the default codec for encoding a base Google Protobuf message or an embedded message by calling the appropriate field codecs for every field 110

goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >
Provides a basic bounded arbitrary length numeric (double, float, uint32, uint64, int32, int64) encoder 111

goby::acomms::DCCLDefaultStringCodec
Provides an variable length ASCII string encoder. Can encode strings up to 255 bytes by using a length byte preceeding the string 113

goby::acomms::DCCLException
Exception class for libdccl 114

goby::acomms::DCCLFieldCodecBase
Provides a base class for defining DCCL field encoders / decoders. Most users will use the DCCLTypedFieldCodec or its children (e.g. DCCLTypedFixedFieldCodec) instead of directly inheriting from this class 114

goby::acomms::DCCLFieldCodecManager 126

goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable >
If WireType == FieldType, we don't have to add any more virtual methods for converting between them 128

goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type >
If WireType != FieldType, adds some more pure virtual methods to handle the type conversions (pre_encode() and post_decode()). If WireType == FieldType this class is not inherited and this pure virtual methods do not exist (and thus can be omitted in the child class) 129

goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >
Base class for "repeated" (multiple value) static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire). Use DCCLTypedFieldCodec if your fields are always singular ("optional" or "required"). Singular fields are default implemented in this codec by calls to the equivalent repeated function with an empty or single valued vector 130

goby::acomms::DCCLStaticCodec< T >
Placeholder codec that takes no space on the wire (0 bits) 134

goby::acomms::DCCLTimeCodec< TimeType >
Encodes time of day (second precision) for times represented by the string representation of boost::posix_time::ptime (e.g. obtained from goby_time<std::string>()) 135

goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >
Base class for static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire) 137

goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType >
Base class for static-typed field encoders/decoders that use a fixed number of bits on the wire regardless of the value of the field. Use DCCLTypedFieldCodec if your encoder is variable length. See DCCLTypedFieldCodec for an explanation of the template parameters (FieldType and WireType) 140

goby::acomms::DCCLTypeHelper
Provides FromProtoTypeBase and FromProtoCppTypeBase type identification helper classes for various representations of the underlying field 141

goby::acomms::FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE >
Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using the dynamic google::protobuf::Message as the underlying class 142

goby::acomms::FromProtoCppTypeBase
Provides various representations of a google::protobuf::FieldDescriptor::CppType enumeration, and ways to access the google::protobuf::Reflection object for a given type 143

goby::acomms::FromProtoCustomMessage< CustomMessage >
Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using a specific statically generated Protobuf class 145

goby::acomms::FromProtoTypeBase
Provides various representations of a google::protobuf::FieldDescriptor::Type enumeration. Implementations are provided for all enumerations 146

goby::acomms::MACManager
API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer 146

goby::acomms::MMDriver
API to the WHOI Micro-Modem driver 148

goby::acomms::ModemDriverBase
Abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems 150

goby::acomms::QueueException
Exception class for libdccl 155

goby::acomms::QueueManager
API to the goby-acomms Queuing Library 156

goby::common::Colors
Represents the eight available terminal colors (and bold variants) 161

goby::common::ConfigException
Indicates a problem with the runtime command line or .cfg file configuration (or –help was given) 162

goby::common::FlexNCurses
Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content 162

goby::common::FlexOstream
Forms the basis of the Goby logger: std::ostream derived class for holding the FlexOStreamBuf 163

goby::common::FlexOStreamBuf
Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by goby::common::FlexOstream for the Goby Logger (glogger) 164

goby::common::TermColor
Converts between string, escape code, and enumeration representations of the terminal colors 165

goby::Exception
Simple exception class for goby applications 166

goby::moos::BluefinCommsDriver
Driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware) 167

goby::moos::UFldDriver
Simulator driver to the uFldNodeComms MOOS module: http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Modules.UFldNodeComms 168

goby::pb::Application
Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture 170

goby::transitional::DCCLMessageVal
Defines a DCCL value 172

goby::transitional::DCCLTransitionalCodec
API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly 176

goby::util::LineBasedInterface
Basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.) 178

goby::util::SerialClient
Basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control 179

goby::util::TCPClient
Basic TCP client for line by line text based communications to a remote TCP server 180

goby::util::TCPServer
Basic TCP server for line by line text based communications to a one or more remote TCP clients 181

Group
Defines a group of messages to be sent to the Goby logger. For Verbosity == verbose streams, all entries appear interleaved, but each group is offset with a different color. For Verbosity == gui streams, all groups have a separate subwindow 183

GroupSetter
Helper class for enabling the group(std::string) manipulator 183

14 File Index

14.1 File List

Here is a list of all documented files with brief descriptions:

goby/acomms.h ??

goby/util.h ??

goby/version.h ??

goby/acomms/acomms_constants.h ??

goby/acomms/acomms_helpers.h ??

goby/acomms/amac.h ??

goby/acomms/bind.h ??

goby/acomms/connect.h ??

goby/acomms/dccl.h ??

goby/acomms/modem_driver.h ??

goby/acomms/queue.h ??

goby/acomms/route.h ??

goby/acomms/amac/mac_manager.h ??

goby/acomms/dccl/dccl.h ??

goby/acomms/dccl/dccl_bitset.h ??

goby/acomms/dccl/dccl_ccl_compatibility.h ??

goby/acomms/dccl/dccl_common.h ??

goby/acomms/dccl/dccl_exception.h ??

goby/acomms/dccl/dccl_field_codec.h ??

goby/acomms/dccl/dccl_field_codec_arithmetic.h ??

goby/acomms/dccl/dccl_field_codec_default.h ??

goby/acomms/dccl/dccl_field_codec_default_message.h ??

goby/acomms/dccl/dccl_field_codec_fixed.h ??

goby/acomms/dccl/dccl_field_codec_helpers.h ??

goby/acomms/dccl/dccl_field_codec_manager.h ??

goby/acomms/dccl/dccl_field_codec_typed.h ??

goby/acomms/dccl/dccl_type_helper.h ??

goby/acomms/dccl/protobuf_cpp_type_helpers.h ??

goby/acomms/dccl/WhoiUtil.h ??

goby/acomms/modemdriver/abc_driver.h ??

goby/acomms/modemdriver/driver_base.h ??

goby/acomms/modemdriver/driver_exception.h ??

goby/acomms/modemdriver/mm_driver.h ??

goby/acomms/modemdriver/udp_driver.h ??

goby/acomms/protobuf/abc_driver.pb.h ??

goby/acomms/protobuf/abc_driver.proto ??

goby/acomms/protobuf/amac.pb.h ??

goby/acomms/protobuf/amac.proto ??

goby/acomms/protobuf/amac_config.pb.h ??

goby/acomms/protobuf/amac_config.proto ??

goby/acomms/protobuf/arithmetic_extensions.pb.h ??

goby/acomms/protobuf/arithmetic_extensions.proto ??

goby/acomms/protobuf/ccl.pb.h ??

goby/acomms/protobuf/ccl.proto ??

goby/acomms/protobuf/ccl_extensions.pb.h ??

goby/acomms/protobuf/ccl_extensions.proto ??

goby/acomms/protobuf/dccl.pb.h ??

goby/acomms/protobuf/dccl.proto ??

goby/acomms/protobuf/dccl_option_extensions.pb.h ??

goby/acomms/protobuf/dccl_option_extensions.proto ??

goby/acomms/protobuf/driver_base.pb.h ??

goby/acomms/protobuf/driver_base.proto ??

goby/acomms/protobuf/file_transfer.pb.h ??

goby/acomms/protobuf/file_transfer.proto ??

goby/acomms/protobuf/manipulator.pb.h ??

goby/acomms/protobuf/manipulator.proto ??

goby/acomms/protobuf/mm_driver.pb.h ??

goby/acomms/protobuf/mm_driver.proto ??

goby/acomms/protobuf/modem_message.pb.h ??

goby/acomms/protobuf/modem_message.proto ??

goby/acomms/protobuf/network_ack.pb.h ??

goby/acomms/protobuf/network_ack.proto ??

goby/acomms/protobuf/queue.pb.h ??

goby/acomms/protobuf/queue.proto ??

goby/acomms/protobuf/route.pb.h ??

goby/acomms/protobuf/route.proto ??

goby/acomms/protobuf/store_server.pb.h ??

goby/acomms/protobuf/store_server.proto ??

goby/acomms/protobuf/udp_driver.pb.h ??

goby/acomms/protobuf/udp_driver.proto ??

goby/acomms/queue/queue.h ??

goby/acomms/queue/queue_constants.h ??

goby/acomms/queue/queue_exception.h ??

goby/acomms/queue/queue_manager.h ??

goby/acomms/route/route.h ??

goby/apps/acomms/goby_bridge/bridge_config.proto ??

goby/apps/acomms/goby_file_transfer/file_transfer_config.proto ??

goby/apps/acomms/goby_modemdriver/modemdriver_config.proto ??

goby/apps/acomms/goby_rudics_shore/goby_rudics_shore_config.proto ??

goby/apps/acomms/goby_store_server/goby_store_server_config.proto ??

goby/apps/common/liaison/liaison.h ??

goby/apps/common/liaison/liaison_home.h ??

goby/apps/common/liaison/liaison_wt_thread.h ??

goby/apps/moos/iFrontSeat/iFrontSeat.h ??

goby/apps/moos/iFrontSeat/legacy_translator.h ??

goby/apps/moos/moos_gateway_g/moos_gateway_config.proto ??

goby/apps/moos/pAcommsHandler/pAcommsHandler.h ??

goby/apps/moos/pTranslator/pTranslator.h ??

goby/apps/moos/pTranslator/pTranslator_config.proto ??

goby/common/application_base.h ??

goby/common/configuration_reader.h ??

goby/common/core_constants.h ??

goby/common/core_helpers.h ??

goby/common/exception.h ??

goby/common/liaison_container.h ??

goby/common/logger.h ??

goby/common/node_interface.h ??

goby/common/pubsub_node_wrapper.h ??

goby/common/time.h ??

goby/common/zeromq_application_base.h ??

goby/common/zeromq_service.h ??

goby/common/logger/flex_ncurses.h ??

goby/common/logger/flex_ostream.h ??

goby/common/logger/flex_ostreambuf.h ??

goby/common/logger/logger_manipulators.h ??

goby/common/logger/term_color.h ??

goby/common/protobuf/app_base_config.pb.h ??

goby/common/protobuf/app_base_config.proto ??

goby/common/protobuf/liaison_config.pb.h ??

goby/common/protobuf/liaison_config.proto ??

goby/common/protobuf/logger.pb.h ??

goby/common/protobuf/logger.proto ??

goby/common/protobuf/option_extensions.pb.h ??

goby/common/protobuf/option_extensions.proto ??

goby/common/protobuf/pubsub_node_config.pb.h ??

goby/common/protobuf/pubsub_node_config.proto ??

goby/common/protobuf/zero_mq_node_config.pb.h ??

goby/common/protobuf/zero_mq_node_config.proto ??

goby/moos/dynamic_moos_vars.h ??

goby/moos/goby_moos_app.h ??

goby/moos/liaison_commander.h ??

goby/moos/liaison_geodesy.h ??

goby/moos/liaison_scope.h ??

goby/moos/modem_id_convert.h ??

goby/moos/moos_bluefin_driver.h ??

goby/moos/moos_geodesy.h ??

goby/moos/moos_header.h ??

goby/moos/moos_liaison_load.h ??

goby/moos/moos_node.h ??

goby/moos/moos_protobuf_helpers.h
Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages 184

goby/moos/moos_serializer.h ??

goby/moos/moos_string.h ??

goby/moos/moos_translator.h ??

goby/moos/moos_ufield_sim_driver.h ??

goby/moos/frontseat/frontseat.h ??

goby/moos/frontseat/frontseat_exception.h ??

goby/moos/frontseat/bluefin/bluefin.h ??

goby/moos/frontseat/bluefin/bluefin.pb.h ??

goby/moos/frontseat/bluefin/bluefin.proto ??

goby/moos/frontseat/bluefin/bluefin_config.pb.h ??

goby/moos/frontseat/bluefin/bluefin_config.proto ??

goby/moos/protobuf/bluefin_driver.pb.h ??

goby/moos/protobuf/bluefin_driver.proto ??

goby/moos/protobuf/ctd_sample.pb.h ??

goby/moos/protobuf/ctd_sample.proto ??

goby/moos/protobuf/desired_course.pb.h ??

goby/moos/protobuf/desired_course.proto ??

goby/moos/protobuf/frontseat.pb.h ??

goby/moos/protobuf/frontseat.proto ??

goby/moos/protobuf/frontseat_config.pb.h ??

goby/moos/protobuf/frontseat_config.proto ??

goby/moos/protobuf/goby_moos_app.pb.h ??

goby/moos/protobuf/goby_moos_app.proto ??

goby/moos/protobuf/liaison_config.pb.h ??

goby/moos/protobuf/liaison_config.proto ??

goby/moos/protobuf/modem_id_lookup.pb.h ??

goby/moos/protobuf/modem_id_lookup.proto ??

goby/moos/protobuf/node_status.pb.h ??

goby/moos/protobuf/node_status.proto ??

goby/moos/protobuf/pAcommsHandler_config.pb.h ??

goby/moos/protobuf/pAcommsHandler_config.proto ??

goby/moos/protobuf/transitional.pb.h ??

goby/moos/protobuf/transitional.proto ??

goby/moos/protobuf/translator.pb.h ??

goby/moos/protobuf/translator.proto ??

goby/moos/protobuf/ufield_sim_driver.pb.h ??

goby/moos/protobuf/ufield_sim_driver.proto ??

goby/moos/transitional/dccl_constants.h ??

goby/moos/transitional/dccl_transitional.h ??

goby/moos/transitional/message.h ??

goby/moos/transitional/message_algorithms.h ??

goby/moos/transitional/message_publish.h ??

goby/moos/transitional/message_val.h ??

goby/moos/transitional/message_var.h ??

goby/moos/transitional/message_var_bool.h ??

goby/moos/transitional/message_var_enum.h ??

goby/moos/transitional/message_var_float.h ??

goby/moos/transitional/message_var_head.h ??

goby/moos/transitional/message_var_hex.h ??

goby/moos/transitional/message_var_int.h ??

goby/moos/transitional/message_var_static.h ??

goby/moos/transitional/message_var_string.h ??

goby/moos/transitional/message_xml_callbacks.h ??

goby/moos/transitional/queue_xml_callbacks.h ??

goby/moos/transitional/xml/message_schema.xsd.h ??

goby/moos/transitional/xml/tags.h ??

goby/moos/transitional/xml/xerces_strings.h ??

goby/moos/transitional/xml/xml_parser.h ??

goby/pb/application.h ??

goby/pb/iridium_driver.h ??

goby/pb/iridium_driver_fsm.h ??

goby/pb/pb_modem_driver.h ??

goby/pb/protobuf_node.h ??

goby/pb/protobuf_pubsub_node_wrapper.h ??

goby/pb/rudics_packet.h ??

goby/pb/subscription.h ??

goby/pb/protobuf/config.pb.h ??

goby/pb/protobuf/config.proto ??

goby/pb/protobuf/database_request.pb.h ??

goby/pb/protobuf/database_request.proto ??

goby/pb/protobuf/header.pb.h ??

goby/pb/protobuf/header.proto ??

goby/pb/protobuf/interprocess_notification.pb.h ??

goby/pb/protobuf/interprocess_notification.proto ??

goby/pb/protobuf/iridium_driver.pb.h ??

goby/pb/protobuf/iridium_driver.proto ??

goby/pb/protobuf/pb_modem_driver.pb.h ??

goby/pb/protobuf/pb_modem_driver.proto ??

goby/pb/protobuf/rudics_shore.pb.h ??

goby/pb/protobuf/rudics_shore.proto ??

goby/test/acomms/dccl1/test.proto ??

goby/test/acomms/dccl10/test.proto ??

goby/test/acomms/dccl2/test.proto ??

goby/test/acomms/dccl3/header.proto ??

goby/test/acomms/dccl3/test.proto ??

goby/test/acomms/dccl4/test.proto ??

goby/test/acomms/dccl6/test.proto ??

goby/test/acomms/dccl7/test.proto ??

goby/test/acomms/dccl8/test.proto ??

goby/test/acomms/dccl9/test.proto ??

goby/test/acomms/driver_tester/driver_tester.h ??

goby/test/acomms/mmdriver2/test_config.proto ??

goby/test/acomms/queue1/test.proto ??

goby/test/acomms/queue5/test.proto ??

goby/test/acomms/queue6/test.proto ??

goby/test/acomms/route1/test.proto ??

goby/test/moos/translator1/basic_node_report.proto ??

goby/test/util/dynamic_protobuf/test_a.proto ??

goby/test/util/dynamic_protobuf/test_b.proto ??

goby/util/as.h ??

goby/util/base_convert.h ??

goby/util/binary.h ??

goby/util/dynamic_protobuf_manager.h ??

goby/util/linebasedcomms.h ??

goby/util/primitive_types.h ??

goby/util/sci.h ??

goby/util/linebasedcomms/client_base.h ??

goby/util/linebasedcomms/connection.h ??

goby/util/linebasedcomms/interface.h ??

goby/util/linebasedcomms/nmea_sentence.h ??

goby/util/linebasedcomms/serial_client.h ??

goby/util/linebasedcomms/tcp_client.h ??

goby/util/linebasedcomms/tcp_server.h ??

goby/util/protobuf/linebasedcomms.pb.h ??

goby/util/protobuf/linebasedcomms.proto ??

goby/util/seawater/depth.h ??

goby/util/seawater/pressure.h ??

goby/util/seawater/salinity.h ??

goby/util/seawater/swstate.h ??

share/examples/acomms/amac/amac_simple/amac_simple.cpp ??

share/examples/acomms/chat/chat.cpp ??

share/examples/acomms/chat/chat.proto ??

share/examples/acomms/chat/chat_curses.cpp ??

share/examples/acomms/chat/chat_curses.h ??

share/examples/acomms/dccl/dccl_simple/dccl_simple.cpp ??

share/examples/acomms/dccl/dccl_simple/simple.proto ??

share/examples/acomms/dccl/two_message/two_message.cpp ??

share/examples/acomms/dccl/two_message/two_message.proto ??

share/examples/acomms/modemdriver/driver_simple/driver_simple.cpp ??

share/examples/acomms/modemdriver/whoi_ranging/whoi_ranging.cpp ??

share/examples/acomms/queue/encode_on_demand/encode_on_demand.cpp ??

share/examples/acomms/queue/encode_on_demand/on_demand.proto ??

share/examples/acomms/queue/multimessage/multimessage.cpp ??

share/examples/acomms/queue/queue_simple/queue_simple.cpp ??

share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.cpp ??

share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.h ??

share/examples/moos/abc_frontseat_driver/abc_frontseat_driver.proto ??

share/examples/moos/abc_frontseat_driver/abc_frontseat_driver_config.proto ??

share/examples/moos/gobyexample_protobuf/simple_status.proto ??

share/examples/util/flexostream_simple/flexostream_simple.cpp ??

src/acomms/amac/mac_manager.cpp ??

src/acomms/dccl/dccl.cpp ??

src/acomms/dccl/dccl_bitset.cpp ??

src/acomms/dccl/dccl_ccl_compatibility.cpp ??

src/acomms/dccl/dccl_field_codec.cpp ??

src/acomms/dccl/dccl_field_codec_arithmetic.cpp ??

src/acomms/dccl/dccl_field_codec_default.cpp ??

src/acomms/dccl/dccl_field_codec_default_message.cpp ??

src/acomms/dccl/dccl_field_codec_helpers.cpp ??

src/acomms/dccl/dccl_field_codec_manager.cpp ??

src/acomms/dccl/dccl_type_helper.cpp ??

src/acomms/dccl/WhoiUtil.cpp ??

src/acomms/modemdriver/abc_driver.cpp ??

src/acomms/modemdriver/driver_base.cpp ??

src/acomms/modemdriver/mm_driver.cpp ??

src/acomms/modemdriver/udp_driver.cpp ??

src/acomms/queue/queue.cpp ??

src/acomms/queue/queue_manager.cpp ??

src/acomms/route/route.cpp ??

src/apps/acomms/abc_modem_simulator/abc_modem_simulator.cpp ??

src/apps/acomms/analyze_dccl/analyze_dccl.cpp ??

src/apps/acomms/goby_bridge/bridge.cpp ??

src/apps/acomms/goby_file_transfer/file_transfer.cpp ??

src/apps/acomms/goby_modemdriver/modemdriver.cpp ??

src/apps/acomms/goby_rudics_shore/goby_rudics_shore.cpp ??

src/apps/acomms/goby_store_server/goby_store_server.cpp ??

src/apps/common/liaison/liaison.cpp ??

src/apps/common/liaison/liaison_home.cpp ??

src/apps/common/liaison/liaison_wt_thread.cpp ??

src/apps/moos/abc_frontseat_simulator/abc_frontseat_simulator.cpp ??

src/apps/moos/dccl_xml_to_dccl_proto/dccl_xml_to_dccl_proto.cpp ??

src/apps/moos/iFrontSeat/iFrontSeat.cpp ??

src/apps/moos/iFrontSeat/legacy_translator.cpp ??

src/apps/moos/moos_gateway_g/moos_gateway.cpp ??

src/apps/moos/pAcommsHandler/pAcommsHandler.cpp ??

src/apps/moos/pAcommsHandler/pAcommsHandlerMain.cpp ??

src/apps/moos/pTranslator/pTranslator.cpp ??

src/apps/moos/pTranslator/pTranslatorMain.cpp ??

src/apps/util/serial2tcp_server/serial2tcp_server.cpp ??

src/common/application_base.cpp ??

src/common/configuration_reader.cpp ??

src/common/time.cpp ??

src/common/zeromq_service.cpp ??

src/common/logger/flex_ncurses.cpp ??

src/common/logger/flex_ostream.cpp ??

src/common/logger/flex_ostreambuf.cpp ??

src/common/logger/logger_manipulators.cpp ??

src/common/logger/term_color.cpp ??

src/moos/goby_moos_app.cpp ??

src/moos/liaison_commander.cpp ??

src/moos/liaison_geodesy.cpp ??

src/moos/liaison_scope.cpp ??

src/moos/modem_id_convert.cpp ??

src/moos/moos_bluefin_driver.cpp ??

src/moos/moos_geodesy.cpp ??

src/moos/moos_liaison_load.cpp ??

src/moos/moos_node.cpp ??

src/moos/moos_translator.cpp ??

src/moos/moos_ufield_sim_driver.cpp ??

src/moos/frontseat/frontseat.cpp ??

src/moos/frontseat/bluefin/bluefin.cpp ??

src/moos/frontseat/bluefin/bluefin_incoming.cpp ??

src/moos/transitional/dccl_transitional.cpp ??

src/moos/transitional/message.cpp ??

src/moos/transitional/message_algorithms.cpp ??

src/moos/transitional/message_publish.cpp ??

src/moos/transitional/message_val.cpp ??

src/moos/transitional/message_var.cpp ??

src/moos/transitional/message_var_float.cpp ??

src/moos/transitional/message_xml_callbacks.cpp ??

src/moos/transitional/queue_xml_callbacks.cpp ??

src/pb/application.cpp ??

src/pb/iridium_driver.cpp ??

src/pb/iridium_driver_fsm.cpp ??

src/pb/pb_modem_driver.cpp ??

src/pb/protobuf_node.cpp ??

src/pb/rudics_packet.cpp ??

src/test/acomms/amac1/test.cpp ??

src/test/acomms/bitset1/test.cpp ??

src/test/acomms/dccl1/test.cpp ??

src/test/acomms/dccl10/test.cpp ??

src/test/acomms/dccl2/test.cpp ??

src/test/acomms/dccl3/test.cpp ??

src/test/acomms/dccl4/test.cpp ??

src/test/acomms/dccl6/test.cpp ??

src/test/acomms/dccl7/test.cpp ??

src/test/acomms/dccl8/test.cpp ??

src/test/acomms/dccl9/test.cpp ??

src/test/acomms/driver_tester/driver_tester.cpp ??

src/test/acomms/mmdriver1/test.cpp ??

src/test/acomms/mmdriver2/test.cpp ??

src/test/acomms/queue1/test.cpp ??

src/test/acomms/queue2/test.cpp ??

src/test/acomms/queue3/test.cpp ??

src/test/acomms/queue4/test.cpp ??

src/test/acomms/queue5/test.cpp ??

src/test/acomms/queue6/test.cpp ??

src/test/acomms/route1/test.cpp ??

src/test/acomms/udpdriver1/test.cpp ??

src/test/acomms/udpdriver2/test.cpp ??

src/test/common/log/test.cpp ??

src/test/common/zero_mq_node1/test.cpp ??

src/test/common/zero_mq_node2/test.cpp ??

src/test/common/zero_mq_node3/test.cpp ??

src/test/common/zero_mq_node4/test.cpp ??

src/test/moos/transitional1/test.cpp ??

src/test/moos/translator1/test.cpp ??

src/test/pb/iridiumdriver1/test.cpp ??

src/test/pb/pbdriver1/test.cpp ??

src/test/util/as/as.cpp ??

src/test/util/base255/base255.cpp ??

src/test/util/dynamic_protobuf/dynamic_protobuf.cpp ??

src/test/util/hex_codec/hex_codec.cpp ??

src/test/util/nmea/nmea.cpp ??

src/test/util/salinity/salinity.cpp ??

src/test/util/sci/sci.cpp ??

src/test/util/time/time.cpp ??

src/util/dynamic_protobuf_manager.cpp ??

src/util/linebasedcomms/interface.cpp ??

src/util/linebasedcomms/nmea_sentence.cpp ??

src/util/linebasedcomms/serial_client.cpp ??

src/util/linebasedcomms/tcp_client.cpp ??

src/util/linebasedcomms/tcp_server.cpp ??

15 Module Documentation

15.1 API classes for the Dynamic Compact Control Language (includes writing custom encoders).

Collaboration diagram for API classes for the Dynamic Compact Control Language (includes writing custom encoders).:

API classes for the Dynamic Compact Control Language (includes writing custom encoders).
goby::acomms::DCCLCodec
API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver).

Classes

- class goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType >

  Base class for static-typed field encoders/decoders that use a fixed number of bits on the wire regardless of the value of the field. Use DCCLTypedFieldCodec if your encoder is variable length. See DCCLTypedFieldCodec for an explanation of the template parameters (FieldType and WireType).
- class goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >

  Base class for static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire).
- class goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >

  Base class for "repeated" (multiple value) static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire). Use DCCLTypedFieldCodec if your fields are always singular ("optional" or "required"). Singular fields are default implemented in this codec by calls to the equivalent repeated function with an empty or single valued vector.
- class goby::acomms::DCCLCodec

  provides an API to the Dynamic CCL Codec.

15.1.1 Detailed Description

15.2 API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver).

Collaboration diagram for API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver).:

API classes for the major components of the Goby-Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver).
goby::acomms::DCCLCodec
API classes for the Dynamic Compact Control Language (includes writing custom encoders).

Classes

- class goby::acomms::ABCDriver

  provides an API to the imaginary ABC modem (as an example how to write drivers)

- class goby::moos::BluefinCommsDriver

  provides a driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware)
- class goby::moos::UFldDriver

  provides an simulator driver to the uFldNodeComms MOOS module: http://oceanai.mit.edu/moos-ivp/pmwiki/pmwiki.php?n=Modules.UFldNodeComms
- class goby::acomms::MACManager

  provides an API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer.
- class goby::acomms::DCCLCodec

  provides an API to the Dynamic CCL Codec.
- class goby::acomms::ModemDriverBase

  provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.
- class goby::acomms::MMDriver

  provides an API to the WHOI Micro-Modem driver
- class goby::acomms::QueueManager

  provides an API to the goby-acomms Queuing Library.
- class goby::transitional::DCCLTransitionalCodec

  provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

15.2.1 Detailed Description

16 Namespace Documentation

16.1 goby Namespace Reference

The global namespace for the Goby project.

Namespaces

- namespace acomms

  Objects pertaining to acoustic communications (acomms)
- namespace common

  Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.
- namespace pb

  Contains objects relating to the core publish / subscribe architecture provided by Goby.
- namespace transitional

  Objects pertaining to transitioning from DCCLv1 to DCCLv2.

Classes

- class Exception

  simple exception class for goby applications

Typedefs

- typedef google::protobuf::uint32 uint32

  an unsigned 32 bit integer
- typedef google::protobuf::int32 int32

  a signed 32 bit integer
- typedef google::protobuf::uint64 uint64

  an unsigned 64 bit integer
- typedef google::protobuf::int64 int64

  a signed 64 bit integer

Enumerations

- enum **GobyFieldOptions_ConfigurationOptions_ConfigAction** { **GobyFieldOptions_ConfigurationOptions_ConfigAction_ALWAYS** = 1, **GobyFieldOptions_ConfigurationOptions_ConfigAction_NEVER** = 2, **GobyFieldOptions_ConfigurationOptions_ConfigAction_ADVANCED** = 3 }

Functions

- template<typename App >
  int run (int argc, char *argv[])

  Run a Goby application derived from MinimalApplicationBase. blocks caller until MinimalApplicationBase::__run() returns.
- void **protobuf_AddDesc_goby_2fcommon_2fprotobuf_2foption_5fextensions_2eproto** ()
- void **protobuf_AssignDesc_goby_2fcommon_2fprotobuf_2foption_5fextensions_2eproto** ()
- void **protobuf_ShutdownFile_goby_2fcommon_2fprotobuf_2foption_5fextensions_2eproto** ()
- bool **GobyFieldOptions_ConfigurationOptions_ConfigAction_IsValid** (int value)
- const ::google::protobuf::EnumDescriptor * **GobyFieldOptions_ConfigurationOptions_ConfigAction_descriptor** ()
- const ::std::string & **GobyFieldOptions_ConfigurationOptions_ConfigAction_Name** (GobyFieldOptions_ConfigurationOptions_ConfigAction value)
- bool **GobyFieldOptions_ConfigurationOptions_ConfigAction_Parse** (const ::std::string &name, GobyFieldOptions_ConfigurationOptions_ConfigAction *value)
- std::string **version_message** ()

Variables

- const GobyFieldOptions_ConfigurationOptions_ConfigAction **GobyFieldOptions_ConfigurationOptions_ConfigAction_ConfigAction_MIN** = GobyFieldOptions_ConfigurationOptions_ConfigAction_ALWAYS
- const GobyFieldOptions_ConfigurationOptions_ConfigAction **GobyFieldOptions_ConfigurationOptions_ConfigAction_ConfigAction_MAX** = GobyFieldOptions_ConfigurationOptions_ConfigAction_ADVANCED
- const int **GobyFieldOptions_ConfigurationOptions_ConfigAction_ConfigAction_ARRAYSIZE** = GobyFieldOptions_ConfigurationOptions_ConfigAction_ConfigAction_MAX + 1
- extern::google::protobuf::internal::ExtensionIdentifier < ::google::protobuf::FieldOptions,::google::protobuf::internal::MessageTypeTraits < ::goby::GobyFieldOptions > , 11, false > **field**
- extern::google::protobuf::internal::ExtensionIdentifier < ::google::protobuf::MessageOptions,::google::protobuf::internal::MessageTypeTraits < ::goby::GobyMessageOptions > , 11, false > **msg**

- const std::string **VERSION_STRING** = "2.0.5~rc1"
- const std::string **VERSION_DATE** = "2014.03.25"

Logger

- common::FlexOstream glog

  Access the Goby logger through this object.

16.1.1 Detailed Description

The global namespace for the Goby project. All objects related to the Goby Underwater Autonomy Project.

16.1.2 Function Documentation

16.1.2.1 template<typename App > int goby::run (int *argc*, char * *argv[]*)

Run a Goby application derived from MinimalApplicationBase. blocks caller until MinimalApplicationBase::__run() returns.

Parameters

argc	same as int main(int argc, char* argv)
argv	same as int main(int argc, char* argv)

Returns

same as int main(int argc, char* argv)

Definition at line 109 of file application_base.h.

16.2 goby::acomms Namespace Reference

Objects pertaining to acoustic communications (acomms)

Classes

- class MACManager

  provides an API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer.
- class DCCLCodec

  provides an API to the Dynamic CCL Codec.
- class DCCLException

  Exception class for libdccl.
- class DCCLFieldCodecBase

  Provides a base class for defining DCCL field encoders / decoders. Most users will use the DCCLTypedFieldCodec or its children (e.g. DCCLTypedFixedFieldCodec) instead of directly inheriting from this class.
- class DCCLDefaultIdentifierCodec

  Provides the default 1 byte or 2 byte DCCL ID codec.
- class DCCLDefaultNumericFieldCodec

  Provides a basic bounded arbitrary length numeric (double, float, uint32, uint64, int32, int64) encoder.
- class DCCLDefaultBoolCodec

  Provides a bool encoder. Uses 1 bit if field is `required`*, 2 bits if* `optional`
- class DCCLDefaultStringCodec

Provides an variable length ASCII string encoder. Can encode strings up to 255 bytes by using a length byte preceeding the string.

- class DCCLDefaultBytesCodec

  Provides an fixed length byte string encoder.

- class DCCLDefaultEnumCodec

  *Provides an enum encoder. This converts the enumeration to an integer (based on the enumeration index (**not** its value) and uses DCCLDefaultNumericFieldCodec to encode the integer.*

- class DCCLTimeCodec

  Encodes time of day (second precision) for times represented by the string representation of boost::posix_time::ptime (e.g. obtained from goby_time<std::string>()).

- class DCCLStaticCodec

  Placeholder codec that takes no space on the wire (0 bits).

- class DCCLDefaultMessageCodec

  Provides the default codec for encoding a base Google Protobuf message or an embedded message by calling the appropriate field codecs for every field.

- class DCCLTypedFixedFieldCodec

  Base class for static-typed field encoders/decoders that use a fixed number of bits on the wire regardless of the value of the field. Use DCCLTypedFieldCodec if your encoder is variable length. See DCCLTypedFieldCodec for an explanation of the template parameters (FieldType and WireType).

- class DCCLFieldCodecManager
- class DCCLFieldCodecSelector

  if WireType == FieldType, we don't have to add any more virtual methods for converting between them.

- class DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type >

  If WireType != FieldType, adds some more pure virtual methods to handle the type conversions (pre_encode() and post_decode()). If WireType == FieldType this class is not inherited and this pure virtual methods do not exist (and thus can be omitted in the child class).

- class DCCLTypedFieldCodec

  Base class for static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire).

- class DCCLRepeatedTypedFieldCodec

  Base class for "repeated" (multiple value) static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire). Use DCCLTypedFieldCodec if your fields are always singular ("optional" or "required"). Singular fields are default implemented in this codec by calls to the equivalent repeated function with an empty or single valued vector.

- class DCCLTypeHelper

  Provides FromProtoTypeBase and FromProtoCppTypeBase type identification helper classes for various representations of the underlying field.

- class FromProtoTypeBase

  Provides various representations of a google::protobuf::FieldDescriptor::Type enumeration. Implementations are provided for all enumerations.

- class FromProtoCppTypeBase

  Provides various representations of a google::protobuf::FieldDescriptor::CppType enumeration, and ways to access the google::protobuf::Reflection object for a given type.

- class FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE >

  Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using the dynamic google::protobuf::Message as the underlying class.

- class FromProtoCustomMessage

  Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using a specific statically generated Protobuf class.

- class ABCDriver

  provides an API to the imaginary ABC modem (as an example how to write drivers)

- class ModemDriverBase

provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.

- class MMDriver

  provides an API to the WHOI Micro-Modem driver
- class QueueException

  Exception class for libdccl.
- class QueueManager

  provides an API to the goby-acomms Queuing Library.

Typedefs

- typedef std::list
  < QueuedMessage >::iterator **messages_it**
- typedef std::multimap
  < unsigned, messages_it >
  ::iterator **waiting_for_ack_it**

Enumerations

- enum { **BYTE_MASK** = 7 }

Functions

- std::ostream & **operator**<< (std::ostream &out, const google::protobuf::Message &msg)
- void bind (ModemDriverBase &driver, QueueManager &queue_manager)

  binds the driver link-layer callbacks to the QueueManager
- void bind (MACManager &mac, ModemDriverBase &driver)

  binds the MAC initiate transmission callback to the driver and the driver parsed message callback to the MAC
- void bind (QueueManager &queue_manager, RouteManager &route_manager)

  creates bindings for a RouteManager to control a particular queue (QueueManager)
- void bind (ModemDriverBase &driver, QueueManager &queue_manager, MACManager &mac)

  bind all three (shortcut to calling the other three bind functions)
- template<typename Signal , typename Slot >
  void connect (Signal *signal, Slot slot)

  connect a signal to a slot (e.g. function pointer)
- template<typename Signal , typename Obj , typename A1 >
  void connect (Signal *signal, Obj *obj, void(Obj::*mem_func)(A1))

  connect a signal to a member function with one argument
- template<typename Signal , typename Obj , typename A1 , typename A2 >
  void connect (Signal *signal, Obj *obj, void(Obj::*mem_func)(A1, A2))

  connect a signal to a member function with two arguments
- template<typename Signal , typename Obj , typename A1 , typename A2 , typename A3 >
  void connect (Signal *signal, Obj *obj, void(Obj::*mem_func)(A1, A2, A3))

  connect a signal to a member function with three arguments
- std::ostream & **operator**<< (std::ostream &os, const DCCLCodec &codec)
- bool **operator==** (const Bitset &a, const Bitset &b)
- bool **operator**< (const Bitset &a, const Bitset &b)
- Bitset **operator&** (const Bitset &b1, const Bitset &b2)
- Bitset **operator|** (const Bitset &b1, const Bitset &b2)
- Bitset **operator^** (const Bitset &b1, const Bitset &b2)
- std::ostream & **operator**<< (std::ostream &os, const Bitset &b)
- unsigned **floor_bits2bytes** (unsigned bits)

- unsigned **ceil_bits2bytes** (unsigned bits)
- std::ostream & **operator<<** (std::ostream &os, const DCCLFieldCodecBase &field_codec)
- DCCLException **type_error** (const std::string &action, const std::type_info &expected, const std::type_info &got)
- std::ostream & **operator<<** (std::ostream &os, const Queue &oq)
- std::ostream & operator<< (std::ostream &out, const QueueManager &d)

  outputs information about all available messages (same as info_all)
- void **serialize_rudics_packet** (std::string bytes, std::string *rudics_pkt)
- void **parse_rudics_packet** (std::string *bytes, std::string rudics_pkt)
- std::string **uint32_to_byte_string** (uint32_t i)
- uint32_t **byte_string_to_uint32** (std::string s)

Acoustic MAC Library callback function type definitions

- std::ostream & **operator<<** (std::ostream &os, const MACManager &mac)

Variables

- const unsigned **BITS_IN_BYTE** = 8
- const unsigned **NIBS_IN_BYTE** = 2
- const int BROADCAST_ID = 0

  special modem id for the broadcast destination - no one is assigned this address. Analogous to 192.168.1.255 on a 192.168.1.0 subnet
- const int QUERY_DESTINATION_ID = -1

  special modem id used internally to goby-acomms for indicating that the MAC layer (`amac`) is agnostic to the next destination. The next destination is thus set by the data provider (typically `queue`).
- const int **QUERY_SOURCE_ID** = -1
- const unsigned char **DCCL_CCL_HEADER** = 32
- const unsigned **MULTIMESSAGE_MASK** = 1 << 7
- const unsigned **BROADCAST_MASK** = 1 << 6
- const unsigned **VAR_ID_MASK** = 0xFF ^ MULTIMESSAGE_MASK ^ BROADCAST_MASK
- const unsigned **USER_FRAME_NEXT_SIZE_BYTES** = 1
- const boost::posix_time::time_duration **ON_DEMAND_SKEW** = boost::posix_time::seconds(1)

16.2.1 Detailed Description

Objects pertaining to acoustic communications (acomms)

16.3 goby::common Namespace Reference

Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.

Namespaces

- namespace tcolor

  Contains functions for adding color to Terminal window streams.

Classes

- class ConfigException

  indicates a problem with the runtime command line or .cfg file configuration (or –help was given)
- class FlexNCurses

  Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.
- class FlexOstream

  Forms the basis of the Goby logger: std::ostream derived class for holding the FlexOStreamBuf.
- class FlexOStreamBuf

  Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by goby::common::FlexOstream for the Goby Logger (glogger)
- struct Colors

  Represents the eight available terminal colors (and bold variants)
- class TermColor

  Converts between string, escape code, and enumeration representations of the terminal colors.

Enumerations

- enum **MarshallingScheme** {
  MARSHALLING_UNKNOWN = 0, **MARSHALLING_CSTR** = 1, **MARSHALLING_PROTOBUF** = 2, **MARSHALLING_CCL** = 3,
  MARSHALLING_MOOS = 4, **MARSHALLING_DCCL** = 5, **MARSHALLING_LCM** = 6, **MARSHALLING_MAX** = 6 }
- enum { **LIAISON_INTERNAL_PUBLISH_SOCKET** = 1, **LIAISON_INTERNAL_SUBSCRIBE_SOCKET** = 2 }

Functions

- Wt::WApplication ∗ **create_wt_application** (const Wt::WEnvironment &env)
- std::ostream & operator<< (std::ostream &out, const google::protobuf::Message &msg)

  provides stream output operator for Google Protocol Buffers Message
- void **merge_app_base_cfg** (AppBaseConfig ∗base_cfg, const boost::program_options::variables_map &var_map)
- std::string **liaison_internal_publish_socket_name** ()
- std::string **liaison_internal_subscribe_socket_name** ()
- std::ostream & **operator**<< (FlexOstream &out, char c)
- std::ostream & **operator**<< (FlexOstream &out, signed char c)
- std::ostream & **operator**<< (FlexOstream &out, unsigned char c)
- std::ostream & **operator**<< (FlexOstream &out, const char ∗s)
- std::ostream & **operator**<< (FlexOstream &out, const signed char ∗s)
- std::ostream & **operator**<< (FlexOstream &out, const unsigned char ∗s)
- double ptime2unix_double (boost::posix_time::ptime given_time)

  convert from boost date_time ptime to the number of seconds (including fractional) since 1/1/1970 0:00 UTC ("UNIX Time")
- boost::posix_time::ptime unix_double2ptime (double given_time)

  convert to boost date_time ptime from the number of seconds (including fractional) since 1/1/1970 0:00 UTC ("UNIX Time"): good to the microsecond
- uint64 ptime2unix_microsec (boost::posix_time::ptime given_time)

  convert from boost date_time ptime to the number of microseconds since 1/1/1970 0:00 UTC ("UNIX Time")
- boost::posix_time::ptime unix_microsec2ptime (uint64 given_time)

  convert to boost date_time ptime from the number of microseconds since 1/1/1970 0:00 UTC ("UNIX Time"): good to the microsecond
- boost::posix_time::ptime **nmea_time2ptime** (const std::string &mt)

Variables

- const int **BITS_IN_UINT32** = 32
- const int **BITS_IN_BYTE** = 8
- const std::string **esc_red** = "\33[31m"
- const std::string **esc_lt_red** = "\33[91m"
- const std::string **esc_green** = "\33[32m"
- const std::string **esc_lt_green** = "\33[92m"
- const std::string **esc_yellow** = "\33[33m"
- const std::string **esc_lt_yellow** = "\33[93m"
- const std::string **esc_blue** = "\33[34m"
- const std::string **esc_lt_blue** = "\33[94m"
- const std::string **esc_magenta** = "\33[35m"
- const std::string **esc_lt_magenta** = "\33[95m"
- const std::string **esc_cyan** = "\33[36m"
- const std::string **esc_lt_cyan** = "\33[96m"
- const std::string **esc_white** = "\33[37m"
- const std::string **esc_lt_white** = "\33[97m"
- const std::string **esc_nocolor** = "\33[0m"

Time

- boost::function0< uint64 > **goby_time_function**
- int **goby_time_warp_factor** = 1
- template<typename ReturnType >
  ReturnType goby_time ()

  Returns current UTC time as a boost::posix_time::ptime.
- template<>
  uint64 goby_time< uint64 > ()

  Returns current UTC time as integer microseconds since 1970-01-01 00:00:00.
- template<>
  double goby_time< double > ()

  Returns current UTC time as seconds and fractional seconds since 1970-01-01 00:00:00.
- template<>
  boost::posix_time::ptime **goby_time**< **boost::posix_time::ptime** > ()
- template<>
  std::string goby_time< std::string > ()

  Returns current UTC time as a human-readable string.
- std::string goby_time_as_string (const boost::posix_time::ptime &t=goby_time())

  Simple string representation of goby_time()
- std::string goby_file_timestamp ()

  ISO string representation of goby_time()
- boost::posix_time::ptime time_t2ptime (std::time_t t)

  convert to ptime from time_t from time.h (whole seconds since UNIX)
- std::time_t ptime2time_t (boost::posix_time::ptime t)

  convert from ptime to time_t from time.h (whole seconds since UNIX)
- double time_duration2double (boost::posix_time::time_duration time_of_day)

  time duration to double number of seconds: good to the microsecond

16.3.1 Detailed Description

Utility objects for performing functions such as logging, non-acoustic communication (ethernet / serial), time, scientific, string manipulation, etc.

16.4 goby::common::tcolor Namespace Reference

Contains functions for adding color to Terminal window streams.

Functions

- std::ostream & add_escape_code (std::ostream &os, const std::string &esc_code)
- std::ostream & red (std::ostream &os)

  All text following this manipulator is red. (e.g. std::cout << red << "text";)
- std::ostream & lt_red (std::ostream &os)

  All text following this manipulator is light red (e.g. std::cout << lt_red << "text";)
- std::ostream & green (std::ostream &os)

  All text following this manipulator is green (e.g. std::cout << green << "text";)
- std::ostream & lt_green (std::ostream &os)

  All text following this manipulator is light green (e.g. std::cout << lt_green << "text";)
- std::ostream & yellow (std::ostream &os)

  All text following this manipulator is yellow (e.g. std::cout << yellow << "text";)
- std::ostream & lt_yellow (std::ostream &os)

  All text following this manipulator is light yellow (e.g. std::cout << lt_yellow << "text";)
- std::ostream & blue (std::ostream &os)

  All text following this manipulator is blue (e.g. std::cout << blue << "text";)
- std::ostream & lt_blue (std::ostream &os)

  All text following this manipulator is light blue (e.g. std::cout << lt_blue << "text";)
- std::ostream & magenta (std::ostream &os)

  All text following this manipulator is magenta (e.g. std::cout << magenta << "text";)
- std::ostream & lt_magenta (std::ostream &os)

  All text following this manipulator is light magenta (e.g. std::cout << lt_magenta << "text";)
- std::ostream & cyan (std::ostream &os)

  All text following this manipulator is cyan (e.g. std::cout << cyan << "text";)
- std::ostream & lt_cyan (std::ostream &os)

  All text following this manipulator is light cyan (e.g. std::cout << lt_cyan << "text";)
- std::ostream & white (std::ostream &os)

  All text following this manipulator is white (e.g. std::cout << white << "text";)
- std::ostream & lt_white (std::ostream &os)

  All text following this manipulator is bright white (e.g. std::cout << lt_white << "text";)
- std::ostream & nocolor (std::ostream &os)

  All text following this manipulator is uncolored (e.g. std::cout << green << "green" << nocolor << "uncolored";)

16.4.1 Detailed Description

Contains functions for adding color to Terminal window streams.

16.4.2 Function Documentation

16.4.2.1 std::ostream & goby::common::tcolor::add_escape_code (std::ostream & *os*, const std::string & *esc_code*)

Append the given escape code to the stream os

Parameters

os	ostream to append to
esc_code	escape code to append (e.g. "\33[31m")

Definition at line 29 of file term_color.cpp.

16.5 goby::pb Namespace Reference

Contains objects relating to the core publish / subscribe architecture provided by Goby.

Classes

- class Application

  Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture.

16.5.1 Detailed Description

Contains objects relating to the core publish / subscribe architecture provided by Goby.

16.6 goby::transitional Namespace Reference

Objects pertaining to transitioning from DCCLv1 to DCCLv2.

Classes

- class DCCLTransitionalCodec

  provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.
- class DCCLMessageVal

  defines a DCCL value

Typedefs

- typedef boost::function< void(DCCLMessageVal &)> AlgFunction1

  boost::function for a function taking a single DCCLMessageVal reference. Used for algorithm callbacks.
- typedef boost::function< void(DCCLMessageVal &, const std::vector < DCCLMessageVal > &)> AlgFunction2

  boost::function for a function taking a dccl::MessageVal reference, and the MessageVal of a second part of the message. Used for algorithm callbacks.

Enumerations

- enum DCCLType { dccl_static, dccl_bool, dccl_int, dccl_float, dccl_enum, dccl_string, dccl_hex }

  Enumeration of DCCL types used for sending messages. dccl_enum and dccl_string primarily map to cpp_string, dccl_bool to cpp_bool, dccl_int to cpp_long, dccl_float to cpp_double.
- enum DCCLCppType { cpp_notype, cpp_bool, cpp_string, cpp_long, cpp_double }

  Enumeration of C++ types used in DCCL.
- enum { **POWER2_BITS_IN_BYTE** = 3 }
- enum { **POWER2_NIBS_IN_BYTE** = 1 }

- enum **DCCLHeaderPart** {
  HEAD_CCL_ID = 0, **HEAD_DCCL_ID** = 1, **HEAD_TIME** = 2, **HEAD_SRC_ID** = 3,
  HEAD_DEST_ID = 4, **HEAD_MULTIMESSAGE_FLAG** = 5, **HEAD_BROADCAST_FLAG** = 6, **HEAD_UNUSED** = 7 }
- enum **DCCLHeaderBits** {
  HEAD_CCL_ID_SIZE = 8, **HEAD_DCCL_ID_SIZE** = 9, **HEAD_TIME_SIZE** = 17, **HEAD_SRC_ID_SIZE** = 5,
  HEAD_DEST_ID_SIZE = 5, **HEAD_FLAG_SIZE** = 1, **HEAD_UNUSED_SIZE** = 2 }

Functions

- unsigned **bits2bytes** (unsigned bits)
- unsigned **bytes2bits** (unsigned bytes)
- unsigned **bytes2nibs** (unsigned bytes)
- unsigned **nibs2bytes** (unsigned nibs)
- std::string **type_to_string** (DCCLType type)
- std::string **type_to_protobuf_type** (DCCLType type)
- std::string **type_to_string** (DCCLCppType type)
- std::string **to_str** (DCCLHeaderPart p)
- template<typename Value >
  std::ostream & operator<< (std::ostream &out, const std::map< std::string, Value > &m)
  use this for displaying a human readable version
- template<typename Value >
  std::ostream & **operator**<< (std::ostream &out, const std::multimap< std::string, Value > &m)
- std::ostream & operator<< (std::ostream &out, const std::set< unsigned > &s)
  use this for displaying a human readable version of this STL object
- std::ostream & operator<< (std::ostream &out, const std::set< std::string > &s)
  use this for displaying a human readable version of this STL object
- std::ostream & **operator**<< (std::ostream &os, const DCCLMessageVal &mv)
- std::ostream & **operator**<< (std::ostream &os, const std::vector< DCCLMessageVal > &vm)

Binary encoding

- bool char_array2hex_string (const unsigned char *c, std::string &s, const unsigned int n)
  converts a char (byte) array into a hex string
- bool hex_string2char_array (unsigned char *c, const std::string &s, const unsigned int n)
  turns a string of hex chars ABCDEF into a character array reading each byte 0xAB,0xCD, 0xEF, etc.
- std::string long2binary_string (unsigned long l, unsigned short bits)
  return a string represented the binary value of `l` for `bits` number of bits which reads MSB -> LSB
- std::string binary_string2hex_string (const std::string &bs)
  converts a binary string ("1000101010101010") into a hex string ("8AAA")
- std::string hex_string2binary_string (const std::string &bs)
  converts a boost::dynamic_bitset (similar to std::bitset but without compile time size requirements) into a hex string
- template<typename T >
  bool hex_string2number (const std::string &s, T &t)
  converts a hex string ("8AAA") into a dynamic_bitset
- template<typename T >
  bool number2hex_string (std::string &s, const T &t, unsigned int width=2)
  converts a decimal number of type T into a hex string
- template<typename T >
  std::string number2hex_string (const T &t, unsigned int width=2)
  converts a decimal number of type T into a hex string assuming success

Variables

- const unsigned **DCCL_NUM_HEADER_BYTES** = 6
- const unsigned **DCCL_NUM_HEADER_PARTS** = 8
- const std::string **DCCL_HEADER_NAMES** []

16.6.1 Detailed Description

Objects pertaining to transitioning from DCCLv1 to DCCLv2.

16.6.2 Typedef Documentation

16.6.2.1 typedef boost::function<void (DCCLMessageVal&)> goby::transitional::AlgFunction1

boost::function for a function taking a single DCCLMessageVal reference. Used for algorithm callbacks.

Think of this as a generalized version of a function pointer (void (*)(DCCLMessageVal&)). See http://www.boost.org/doc/libs/1_34_0/doc/html/function.html for more on boost:function.

Definition at line 41 of file message_algorithms.h.

16.6.2.2 typedef boost::function<void (DCCLMessageVal&, const std::vector<DCCLMessageVal>&)> goby::transitional::AlgFunction2

boost::function for a function taking a dccl::MessageVal reference, and the MessageVal of a second part of the message. Used for algorithm callbacks.

Think of this as a generalized version of a function pointer (void (*)(DCCLMessageVal&, const DCCLMessageVal&). See http://www.boost.org/doc/libs/1_34_0/doc/html/function.html for more on boost:function.

Definition at line 50 of file message_algorithms.h.

16.6.3 Enumeration Type Documentation

16.6.3.1 enum goby::transitional::DCCLCppType

Enumeration of C++ types used in DCCL.

Enumerator:

cpp_notype not one of the C++ types used in DCCL

cpp_bool C++ bool

cpp_string C++ std::string

cpp_long C++ long

cpp_double C++ double

Definition at line 59 of file dccl_constants.h.

16.6.3.2 enum goby::transitional::DCCLType

Enumeration of DCCL types used for sending messages. dccl_enum and dccl_string primarily map to cpp_string, dccl_bool to cpp_bool, dccl_int to cpp_long, dccl_float to cpp_double.

Enumerator:

dccl_static tag_static

dccl_bool tag_bool

dccl_int tag_int

dccl_float tag_float

dccl_enum tag_enum

dccl_string tag_string

dccl_hex tag_hex

Definition at line 50 of file dccl_constants.h.

16.6.4 Function Documentation

16.6.4.1 bool goby::transitional::char_array2hex_string (const unsigned char * c, std::string & s, const unsigned int n) [inline]

converts a char (byte) array into a hex string

Parameters

c	pointer to array of char
s	reference to string to put char into as hex
n	length of c the first two hex chars in s are the 0 index in c

Definition at line 173 of file dccl_constants.h.

16.6.4.2 std::string goby::transitional::hex_string2binary_string (const std::string & bs) [inline]

converts a boost::dynamic_bitset (similar to std::bitset but without compile time size requirements) into a hex string

converts a hex string ("8AAA") into a binary string ("1000101010101010")

only works on whole byte string (even number of nibbles)

Definition at line 244 of file dccl_constants.h.

16.6.4.3 template<typename T > bool goby::transitional::hex_string2number (const std::string & s, T & t)

converts a hex string ("8AAA") into a dynamic_bitset

attempts to convert a hex string into a numerical representation (of type T)

Returns

true if conversion succeeds, false otherwise

Definition at line 275 of file dccl_constants.h.

16.6.4.4 template<typename T > bool goby::transitional::number2hex_string (std::string & s, const T & t, unsigned int width = 2)

converts a decimal number of type T into a hex string

Parameters

s	string reference to store result in
t	decimal number to convert
width	desired width (in characters) of return string. Width should be twice the number of bytes

Returns

true if successful, false otherwise

Definition at line 290 of file dccl_constants.h.

16.6.4.5 template<typename T > std::string goby::transitional::number2hex_string (const T & t, unsigned int width = 2)

converts a decimal number of type T into a hex string assuming success

Parameters

t	decimal number to convert
width	desired width (in characters) of return string. Width should be twice the number of bytes

Returns

hex string

Definition at line 303 of file dccl_constants.h.

16.6.5 Variable Documentation

16.6.5.1 const std::string goby::transitional::DCCL_HEADER_NAMES[ ]

Initial value:

```
{ "_ccl_id",
                                                "_id",
                                                "_time",
                                                "_src_id",
                                                "_dest_id",
                                                "_multimessage_flag",
                                                "_broadcast_flag",
                                                "_unused",
        }
```

Definition at line 137 of file dccl_constants.h.

17 Class Documentation

17.1 boost::asio::time_traits< goby::common::GobyTime > Struct Template Reference

Time traits specialised for GobyTime.

`#include <goby/common/time.h>`

Public Types

- typedef boost::posix_time::ptime time_type
  The time type.
- typedef boost::posix_time::time_duration duration_type
  The duration type.

Static Public Member Functions

- static time_type now ()
  Get the current time.
- static time_type add (const time_type &t, const duration_type &d)
  Add a duration to a time.
- static duration_type subtract (const time_type &t1, const time_type &t2)
  Subtract one time from another.
- static bool less_than (const time_type &t1, const time_type &t2)
  Test whether one time is less than another.
- static boost::posix_time::time_duration to_posix_duration (const duration_type &d)
  Convert to POSIX duration type.

17.1.1 Detailed Description

template<>struct boost::asio::time_traits< goby::common::GobyTime >

Time traits specialised for GobyTime.

Definition at line 231 of file time.h.

The documentation for this struct was generated from the following file:

- goby/common/time.h

17.2 ChatCurses Class Reference

provides a terminal GUI for a chat window (lower box to type and upper box to receive messages). Part of the chat.cpp example.

`#include </build/buildd/goby2-2.1.0~beta8+DAILYDEB+536/share/examples/acomms/chat/chat_curses.h>`

Public Member Functions

- void set_modem_id (unsigned id)
  give the modem_id so we know how to label our messages
- void startup ()
  start the display
- void run_input (std::string &line)
  grab a character and if there's a line to return it will be returned in `line`
- void cleanup ()
  end the display
- void post_message (unsigned id, const std::string &line)
  add a message to the upper window (the chat log)
- void **post_message** (const std::string &line)

Constructors/Destructor

- **ChatCurses** ()
- ~**ChatCurses** ()

17.2.1 Detailed Description

provides a terminal GUI for a chat window (lower box to type and upper box to receive messages). Part of the chat.cpp example.

Examples:

acomms/chat/chat.cpp.

Definition at line 9 of file chat_curses.h.

The documentation for this class was generated from the following files:

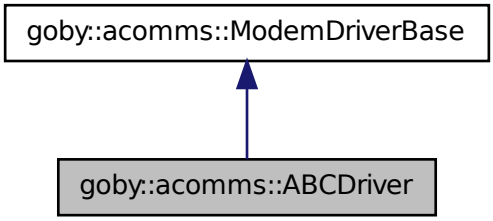
- share/examples/acomms/chat/chat_curses.h
- share/examples/acomms/chat/chat_curses.cpp

17.3 goby::acomms::ABCDriver Class Reference

provides an API to the imaginary ABC modem (as an example how to write drivers)

`#include <goby/acomms/modemdriver/abc_driver.h>`

Inheritance diagram for goby::acomms::ABCDriver:



Public Member Functions

- void startup (const protobuf::DriverConfig &cfg)

  Starts the modem driver. Must be called before poll().
- void shutdown ()

  Shuts down the modem driver.
- void do_work ()

  Allows the modem driver to do its work.
- void handle_initiate_transmission (const protobuf::ModemTransmission &m)

  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Additional Inherited Members

17.3.1 Detailed Description

provides an API to the imaginary ABC modem (as an example how to write drivers)

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 41 of file abc_driver.h.

17.3.2 Member Function Documentation

17.3.2.1 void goby::acomms::ABCDriver::do_work ( ) [virtual]

Allows the modem driver to do its work.

Should be called regularly to perform the work of the driver as the driver *does not* run in its own thread. This allows us to guarantee that no signals are called except inside this method. Does not block.

Implements goby::acomms::ModemDriverBase.

Definition at line 116 of file abc_driver.cpp.

17.3.2.2 void goby::acomms::ABCDriver::handle_initiate_transmission (const protobuf::ModemTransmission & *m*) [virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

m	ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)

Implements goby::acomms::ModemDriverBase.

Definition at line 79 of file abc_driver.cpp.

17.3.2.3 void goby::acomms::ABCDriver::startup (const protobuf::DriverConfig & *cfg*) [virtual]

Starts the modem driver. Must be called before poll().

Parameters

cfg	Startup configuration for the driver and modem. DriverConfig is defined in acomms_driver_base.proto. Derived classes can define extensions (see http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions) to DriverConfig to handle modem specific configuration.

Implements goby::acomms::ModemDriverBase.

Definition at line 41 of file abc_driver.cpp.

The documentation for this class was generated from the following files:

- goby/acomms/modemdriver/abc_driver.h
- src/acomms/modemdriver/abc_driver.cpp

17.4 goby::acomms::DCCLCodec Class Reference

provides an API to the Dynamic CCL Codec.

`#include <goby/acomms/dccl.h>`

Public Member Functions

Initialization Methods.

- void set_cfg (const protobuf::DCCLConfig &cfg)

  Set (and overwrite completely if present) the current configuration. (protobuf::DCCLConfig defined in acomms_dccl.proto)
- void merge_cfg (const protobuf::DCCLConfig &cfg)

  Set (and merge "repeated" fields) the current configuration. (protobuf::DCCLConfig defined in acomms_dccl.proto). Non-repeated fields will be overwritten if set.
- void load_shared_library_codecs (void *dl_handle)

  Load any codecs present in the given shared library handle.
- template<typename ProtobufMessage >
  void validate ()

  All messages must be validated (size checks, option extensions checks, etc.) before they can be encoded/decoded. Use this form when the messages used are static (known at compile time).

Codec functions.

This is where the real work happens.

- void encode (std::string *bytes, const google::protobuf::Message &msg, bool header_only=false)
  Encodes a DCCL message.
- void decode (const std::string &bytes, google::protobuf::Message *msg, bool header_only=false)
  Decode a DCCL message when the type is known at compile time.
- unsigned id_from_encoded (const std::string &bytes)
  Get the DCCL ID of an unknown encoded DCCL message.

Alternative Dynamic Protobuf methods (including list of Message methods) (Advanced)

These methods are intended to be used when the Google Protobuf message types are not known at compile-time, and work with the `Descriptor`/`Reflection` *meta-data introspection API provided by Google.*

- void validate (const google::protobuf::Descriptor *desc)
  An alterative form for validating messages for message types not known at compile-time ("dynamic").
- void validate_repeated (const std::list< const google::protobuf::Descriptor * > &descs)
  Shortcut for validating multiple messages at once.
- void info (const google::protobuf::Descriptor *desc, std::ostream *os) const
  An alterative form for getting information for messages for message types not known at compile-time ("dynamic").
- void info_repeated (const std::list< const google::protobuf::Descriptor * > &desc, std::ostream *os) const
  Shortcut for getting information on multiple messages at once.
- unsigned id (const google::protobuf::Descriptor *desc) const
  Provides the DCCL ID given a DCCL type.
- template<typename GoogleProtobufMessagePointer >
  unsigned size_repeated (const std::list< GoogleProtobufMessagePointer > &msgs)
  Returns the encoded size of multiple messages at once (sum of all sizes). You must use this method (instead of N calls to size, where N is msgs.size()) to get the size of messages encoded using encode_repeated as repeated encoding may be more efficient (in terms of the encoded message size) than N calls of encode.
- template<typename GoogleProtobufMessagePointer >
  GoogleProtobufMessagePointer decode (const std::string &bytes, bool header_only=false)
  An alterative form for decoding messages for message types not known at compile-time ("dynamic").
- template<typename GoogleProtobufMessagePointer >
  std::string encode_repeated (const std::list< GoogleProtobufMessagePointer > &msgs)
  Encode multiple messages at once. In general this is more efficient (i.e. produces smaller messages) than calling encode repeatedly.
- template<typename GoogleProtobufMessagePointer >
  std::list
  < GoogleProtobufMessagePointer > decode_repeated (const std::string &orig_bytes)
  Decode multiple messages at once. Messages encoded using encode_repeated must be decoded using decode_repeated.

Custom DCCL ID Codecs (Advanced)

Change the underlying DCCL ID codec to optimize your network. Be very careful you know how to identify all your messages though!

- template<typename DCCLTypedFieldCodecUint32 >
  void add_id_codec (const std::string &identifier)
  Adds a DCCL id codec to be used along with a string identifier.
- void set_id_codec (const std::string &identifier)
  Sets the DCCL id codec currently in use.
- void reset_id_codec ()
  Resets the DCCL id codec currently in use to the default.
- const std::set< unsigned > & **skip_crypto_ids** ()

Static Public Member Functions

- static DCCLCodec * get ()

  DCCLCodec is a singleton class; use this to get a pointer to the class.

Static Public Attributes

- static const std::string **DEFAULT_CODEC_NAME** = ""

Friends

- template<typename T >
  void **boost::checked_delete** (T *)

Informational Methods.

Provides various forms of information about the DCCLCodec

- static const std::string & glog_encode_group ()

  The group name for goby::glog that all encoding related messages are written.
- static const std::string & glog_decode_group ()

  The group name for goby::glog that all decoding related messages are written.
- template<typename ProtobufMessage >
  void info (std::ostream *os) const

  Writes a human readable summary (including field sizes) of the provided DCCL type to the stream provided.
- void info_all (std::ostream *os) const

  Writes a human readable summary (including field sizes) of all the loaded (validated) DCCL types.
- template<typename ProtobufMessage >
  unsigned id () const

  Gives the DCCL id (defined by the custom message option extension "(goby.msg).dccl.id" in the .proto file). This ID is used on the wire to unique identify incoming message types.
- unsigned size (const google::protobuf::Message &msg)

  Provides the encoded size (in bytes) of msg. This is useful if you need to know the size of a message before encoding it (encoding it is generally much more expensive than calling this method)

17.4.1 Detailed Description

provides an API to the Dynamic CCL Codec.

```
\sa acomms_dccl.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages

Simple usage example:
1. Define a Google Protobuf message with DCCL extensions:
\verbinclude simple.proto
2. Write a bit of code like this:
@code
```

goby::acomms::DCCLCodec* dccl = goby::acomms::DCCLCodec::get(); dccl->validate<Simple>(); Simple message_out; message_out.set_telegram("Hello!"); std::string bytes; dccl->encode(&bytes, message); \ send bytes across some network Simple message_in; dccl->decode(bytes&, message_in);

Examples:

acomms/chat/chat.cpp, acomms/dccl/dccl_simple/dccl_simple.cpp, and acomms/dccl/two_message/two_message.cpp.

Definition at line 96 of file dccl.h.

17.4.2 Member Function Documentation

17.4.2.1 template<typename DCCLTypedFieldCodecUint32 > void goby::acomms::DCCLCodec::add_id_codec (const std::string & *identifier*) [inline]

Adds a DCCL id codec to be used along with a string identifier.

Template Parameters

DCCLTypedFieldCodec-Uint32	Subclass of DCCLTypedFieldCodec<uint32> > that implements encoding / decoding of DCCL IDs

Parameters

identifier	a name to give this DCCL ID codec for later use when setting it.

Definition at line 360 of file dccl.h.

17.4.2.2 void goby::acomms::DCCLCodec::decode (const std::string & *bytes*, google::protobuf::Message * *msg*, bool *header_only* = false)

Decode a DCCL message when the type is known at compile time.

Parameters

bytes	encoded message to decode (must already have been validated)
msg	Pointer to any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::Message). The decoded message will be written here.

Exceptions

DCCLException	if message cannot be decoded.

Examples:

acomms/dccl/dccl_simple/dccl_simple.cpp, and acomms/dccl/two_message/two_message.cpp.

Definition at line 215 of file dccl.cpp.

17.4.2.3 template<typename GoogleProtobufMessagePointer > GoogleProtobufMessagePointer goby::acomms::DCCLCodec::decode (const std::string & *bytes*, bool *header_only* = false) [inline]

An alterative form for decoding messages for message types *not* known at compile-time ("dynamic").

Template Parameters

GoogleProtobufMessage-Pointer	anything that acts like a pointer (has operator*) to a google::protobuf::Message (smart pointers like boost::shared_ptr included)

Parameters

bytes	the byte string returned by encode

Exceptions

DCCLException	if message cannot be decoded

Returns

pointer to decoded message (a google::protobuf::Message). You are responsible for deleting the memory used by this pointer. This message can be examined using the Google Reflection/Descriptor API.

Definition at line 274 of file dccl.h.

17.4.2.4 template<typename GoogleProtobufMessagePointer > std::list<GoogleProtobufMessagePointer> goby::acomms::DCCLCodec::decode_repeated (const std::string & *orig_bytes*) [inline]

Decode multiple messages at once. Messages encoded using encode_repeated must be decoded using decode_-repeated.

Parameters

orig_bytes	string of bytes produced by encoded_repeated

Template Parameters

GoogleProtobufMessage-Pointer	anything that acts like a pointer (has operator∗) to a google::protobuf::Message (smart pointers like boost::shared_ptr included)

Exceptions

DCCLException	if message(s) cannot be decoded.

Returns

list of pointers to the decoded messages: you are responsible for deleting these pointers when done with them.

See Also

test/acomms/dccl4/test.cpp and test/acomms/dccl4/test.proto

Definition at line 318 of file dccl.h.

17.4.2.5 void goby::acomms::DCCLCodec::encode (std::string ∗ *bytes*, const google::protobuf::Message & *msg*, bool *header_only* = false)

Encodes a DCCL message.

Parameters

bytes	Pointer to byte string to store encoded msg
msg	Message to encode (must already have been validated)

Exceptions

DCCLException	if message cannot be encoded.

Examples:

acomms/dccl/dccl_simple/dccl_simple.cpp, and acomms/dccl/two_message/two_message.cpp.

Definition at line 112 of file dccl.cpp.

17.4.2.6 template<typename GoogleProtobufMessagePointer > std::string goby::acomms::DCCLCodec::encode_repeated (const std::list< GoogleProtobufMessagePointer > & *msgs*) [inline]

Encode multiple messages at once. In general this is more efficient (i.e. produces smaller messages) than calling encode repeatedly.

Template Parameters

GoogleProtobufMessage-Pointer	anything that acts like a pointer (has operator*) to a google::protobuf::Message (smart pointers like boost::shared_ptr included)

Parameters

msgs	list of DCCL messages to encode

Exceptions

DCCLException	if message(s) cannot be encoded.

Returns

string containing bytes that represent the encoded message

See Also

test/acomms/dccl4/test.cpp and test/acomms/dccl4/test.proto

Definition at line 297 of file dccl.h.

17.4.2.7 template<typename ProtobufMessage > unsigned goby::acomms::DCCLCodec::id ( ) const [inline]

Gives the DCCL id (defined by the custom message option extension "(goby.msg).dccl.id" in the .proto file). This ID is used on the wire to unique identify incoming message types.

Template Parameters

ProtobufMessage	Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::-Message)

Definition at line 164 of file dccl.h.

17.4.2.8 unsigned goby::acomms::DCCLCodec::id_from_encoded (const std::string & *bytes*)

Get the DCCL ID of an unknown encoded DCCL message.

```
You can use this method along with id() to handle multiple types of known (static) incoming DCCL me
@code
```

unsigned dccl_id = codec->id_from_encoded(bytes); if(dccl_id == codec->id<MyProtobufType1>()) { MyProtobuf-Type1 msg_out1; codec->decode(bytes, &msg_out1); } else if(dccl_id == codec->id<MyProtobufType2>()) { My-ProtobufType2 msg_out2; codec->decode(bytes, &msg_out2); }

Parameters

bytes	encoded message to get the DCCL ID of

Returns

DCCL ID

See Also

test/acomms/dccl8/test.cpp and test/acomms/dccl8/test.proto

Definition at line 197 of file dccl.cpp.

17.4.2.9 template<typename ProtobufMessage > void goby::acomms::DCCLCodec::info (std::ostream * *os*) const [inline]

Writes a human readable summary (including field sizes) of the provided DCCL type to the stream provided.

Template Parameters

ProtobufMessage	Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::Message)

Parameters

os	Pointer to a stream to write this information

Definition at line 152 of file dccl.h.

17.4.2.10 void goby::acomms::DCCLCodec::info_all (std::ostream * *os*) const

Writes a human readable summary (including field sizes) of all the loaded (validated) DCCL types.

Parameters

os	Pointer to a stream to write this information

Definition at line 533 of file dccl.cpp.

17.4.2.11 void goby::acomms::DCCLCodec::load_shared_library_codecs (void * *dl_handle*)

Load any codecs present in the given shared library handle.

Codecs must be loaded within the shared library using a C function (declared extern "C") called "goby_dccl_load" with the signature void goby_dccl_load(goby::acomms::DCCLCodec* codec)

Definition at line 492 of file dccl.cpp.

17.4.2.12 void goby::acomms::DCCLCodec::set_id_codec (const std::string & *identifier*) [inline]

Sets the DCCL id codec currently in use.

The DCCL ID codec will be set and used for all relevant calls to encode, decode, etc. until this method is called again with a different identifier or reset_id_codec is called.

Parameters

identifier	name of this special DCCL ID codec that matches the identifier used when registering with add_id_codec().

Definition at line 370 of file dccl.h.

17.4.2.13 unsigned goby::acomms::DCCLCodec::size (const google::protobuf::Message & *msg*)

Provides the encoded size (in bytes) of msg. This is useful if you need to know the size of a message before encoding it (encoding it is generally much more expensive than calling this method)

Parameters

msg	Google Protobuf message with DCCL extensions for which the encoded size is requested

Returns

Encoded (using DCCL) size in bytes

Definition at line 361 of file dccl.cpp.

17.4.2.14 template<typename GoogleProtobufMessagePointer > unsigned goby::acomms::DCCLCodec::size_repeated (const std::list< GoogleProtobufMessagePointer > & *msgs*) [inline]

Returns the encoded size of multiple messages at once (sum of all sizes). You must use this method (instead of N calls to size, where N is msgs.size()) to get the size of messages encoded using encode_repeated as repeated encoding may be more efficient (in terms of the encoded message size) than N calls of encode.

Template Parameters

GoogleProtobufMessage-Pointer	anything that acts like a pointer (has operator*) to a google::protobuf::Message (smart pointers like boost::shared_ptr included)

Parameters

msgs	list of DCCL messages to get the sizes of

Returns

total size of the encoded messages in bytes

Definition at line 259 of file dccl.h.

17.4.2.15 template<typename ProtobufMessage > void goby::acomms::DCCLCodec::validate ( ) [inline]

All messages must be validated (size checks, option extensions checks, etc.) before they can be encoded/decoded. Use this form when the messages used are static (known at compile time).

Template Parameters

ProtobufMessage	Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::-Message)

Exceptions

DCCLException	if message is invalid. Warnings and errors are written to goby::glog.

Examples:

acomms/chat/chat.cpp, acomms/dccl/dccl_simple/dccl_simple.cpp, and acomms/dccl/two_message/two_-message.cpp.

Definition at line 135 of file dccl.h.

17.4.2.16 void goby::acomms::DCCLCodec::validate (const google::protobuf::Descriptor * *desc*)

An alterative form for validating messages for message types *not* known at compile-time ("dynamic").

Parameters

desc	The Google Protobuf "Descriptor" (meta-data) of the message to validate.

Exceptions

DCCLException	if message is invalid.

Definition at line 311 of file dccl.cpp.

17.4.2.17 void goby::acomms::DCCLCodec::validate_repeated (const std::list< const google::protobuf::Descriptor * > & *descs*)

Shortcut for validating multiple messages at once.

Exceptions

DCCLException	if message(s) are invalid.

Parameters

descs	list of Google Protobuf "Descriptors" to validate

See Also

test/acomms/dccl4/test.cpp and test/acomms/dccl4/test.proto

Definition at line 424 of file dccl.cpp.

The documentation for this class was generated from the following files:

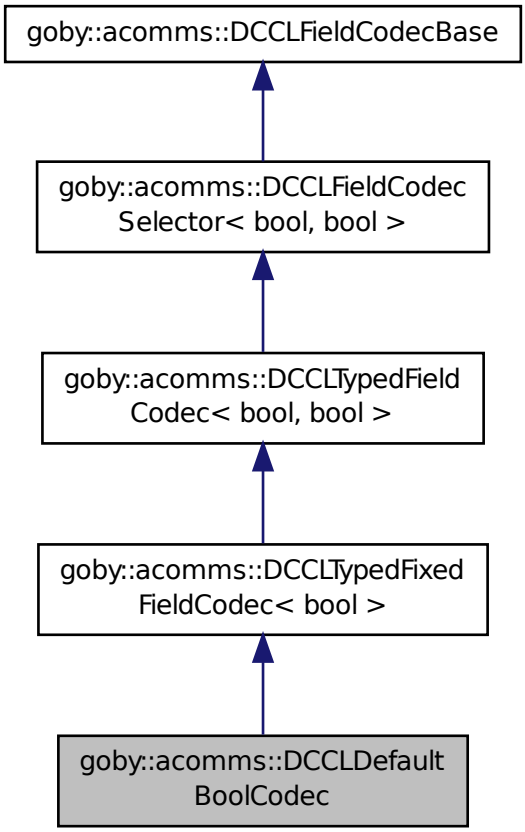
- goby/acomms/dccl/dccl.h
- src/acomms/dccl/dccl.cpp

17.5 goby::acomms::DCCLDefaultBoolCodec Class Reference

Provides a bool encoder. Uses 1 bit if field is `required`, 2 bits if `optional`

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLDefaultBoolCodec:



Additional Inherited Members

17.5.1 Detailed Description

Provides a bool encoder. Uses 1 bit if field is `required`, 2 bits if `optional`

[presence bit (0 bits if required, 1 bit if optional)][value (1 bit)]

Definition at line 171 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following files:

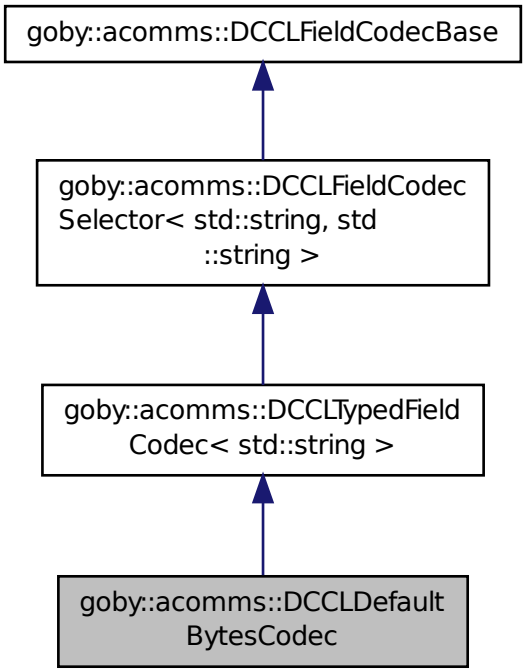
- goby/acomms/dccl/dccl_field_codec_default.h
- src/acomms/dccl/dccl_field_codec_default.cpp

17.6 goby::acomms::DCCLDefaultBytesCodec Class Reference

Provides an fixed length byte string encoder.

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLDefaultBytesCodec:



Additional Inherited Members

17.6.1 Detailed Description

Provides an fixed length byte string encoder.

Definition at line 202 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following files:

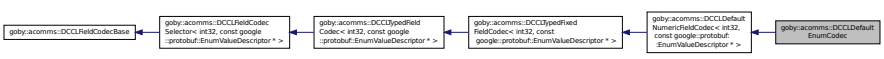
- goby/acomms/dccl/dccl_field_codec_default.h
- src/acomms/dccl/dccl_field_codec_default.cpp

17.7 goby::acomms::DCCLDefaultEnumCodec Class Reference

Provides an enum encoder. This converts the enumeration to an integer (based on the enumeration *index* (**not** its *value*) and uses DCCLDefaultNumericFieldCodec to encode the integer.

```
#include <goby/acomms/dccl/dccl_field_codec_default.h>
```

Inheritance diagram for goby::acomms::DCCLDefaultEnumCodec:



Public Member Functions

- int32 **pre_encode** (const google::protobuf::EnumValueDescriptor ∗const &field_value)
- const google::protobuf::EnumValueDescriptor ∗ **post_decode** (const int32 &wire_value)

Additional Inherited Members

17.7.1 Detailed Description

Provides an enum encoder. This converts the enumeration to an integer (based on the enumeration *index* (**not** its *value*) and uses DCCLDefaultNumericFieldCodec to encode the integer.

Definition at line 216 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following files:

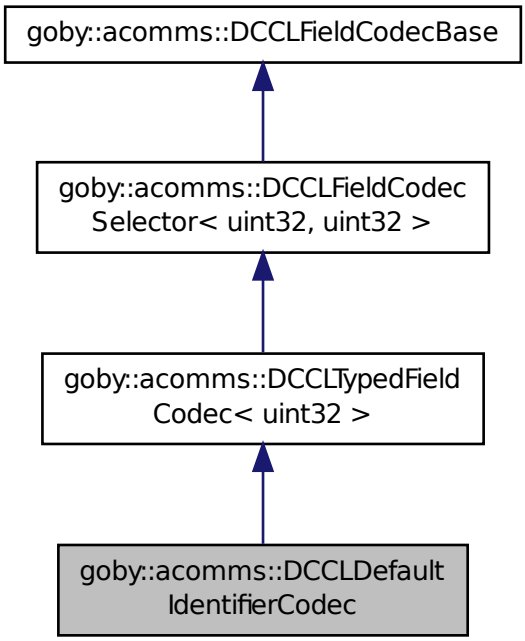
- goby/acomms/dccl/dccl_field_codec_default.h
- src/acomms/dccl/dccl_field_codec_default.cpp

17.8 goby::acomms::DCCLDefaultIdentifierCodec Class Reference

Provides the default 1 byte or 2 byte DCCL ID codec.

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLDefaultIdentifierCodec:



Protected Member Functions

- virtual Bitset encode ()

  Encode an empty field.
- virtual Bitset encode (const uint32 &wire_value)

  Encode a non-empty field.
- virtual uint32 decode (Bitset *bits)

  Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValue-Exception to indicate this.
- virtual unsigned size ()

  Calculate the size (in bits) of an empty field.
- virtual unsigned size (const uint32 &wire_value)

  Calculate the size (in bits) of a non-empty field.
- virtual unsigned max_size ()

  Calculate maximum size of the field in bits.
- virtual unsigned min_size ()

  Calculate minimum size of the field in bits.
- virtual void validate ()

  Validate a field. Use require() inside your overloaded validate() to assert requirements or throw DCCLExceptions directly as needed.

Additional Inherited Members

17.8.1 Detailed Description

Provides the default 1 byte or 2 byte DCCL ID codec.

Definition at line 55 of file dccl_field_codec_default.h.

17.8.2 Member Function Documentation

17.8.2.1 goby::uint32 goby::acomms::DCCLDefaultIdentifierCodec::decode (Bitset * *bits*) [protected], [virtual]

Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValue-Exception to indicate this.

Parameters

bits	Bits to use for decoding.

Returns

the decoded value.

Implements goby::acomms::DCCLTypedFieldCodec< uint32 >.

Definition at line 63 of file dccl_field_codec_default.cpp.

17.8.2.2 goby::acomms::Bitset goby::acomms::DCCLDefaultIdentifierCodec::encode ( ) [protected], [virtual]

Encode an empty field.

Returns

Bits represented the encoded field.

Implements goby::acomms::DCCLTypedFieldCodec< uint32 >.

Definition at line 40 of file dccl_field_codec_default.cpp.

17.8.2.3 goby::acomms::Bitset goby::acomms::DCCLDefaultIdentifierCodec::encode (const uint32 & *wire_value*) [protected], [virtual]

Encode a non-empty field.

Parameters

wire_value	Value to encode.

Returns

Bits represented the encoded field.

Implements goby::acomms::DCCLTypedFieldCodec< uint32 >.

Definition at line 45 of file dccl_field_codec_default.cpp.

17.8.2.4 unsigned goby::acomms::DCCLDefaultIdentifierCodec::max_size ( ) [protected], [virtual]

Calculate maximum size of the field in bits.

Returns

Maximum size of this field (in bits).

Implements goby::acomms::DCCLFieldCodecBase.

Definition at line 103 of file dccl_field_codec_default.cpp.

17.8.2.5 unsigned goby::acomms::DCCLDefaultIdentifierCodec::min_size ( ) [protected], [virtual]

Calculate minimum size of the field in bits.

Returns

Minimum size of this field (in bits).

Implements goby::acomms::DCCLFieldCodecBase.

Definition at line 108 of file dccl_field_codec_default.cpp.

17.8.2.6 unsigned goby::acomms::DCCLDefaultIdentifierCodec::size ( ) [protected], [virtual]

Calculate the size (in bits) of an empty field.

Returns

the size (in bits) of the empty field.

Implements goby::acomms::DCCLTypedFieldCodec< uint32 >.

Definition at line 82 of file dccl_field_codec_default.cpp.

17.8.2.7 unsigned goby::acomms::DCCLDefaultIdentifierCodec::size (const uint32 & *wire_value*) [protected], [virtual]

Calculate the size (in bits) of a non-empty field.

Parameters

wire_value	Value to use when calculating the size of the field. If calculating the size requires encoding the field completely, cache the encoded value for a likely future call to encode() for the same wire_value.

Returns

the size (in bits) of the field.

Implements goby::acomms::DCCLTypedFieldCodec< uint32 >.

Definition at line 87 of file dccl_field_codec_default.cpp.

The documentation for this class was generated from the following files:

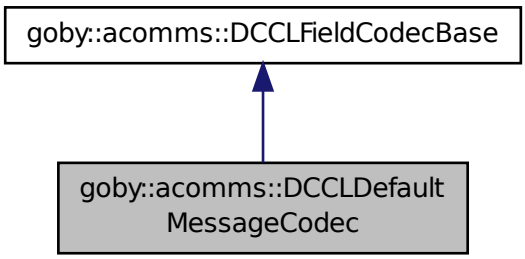
- goby/acomms/dccl/dccl_field_codec_default.h
- src/acomms/dccl/dccl_field_codec_default.cpp

17.9 goby::acomms::DCCLDefaultMessageCodec Class Reference

Provides the default codec for encoding a base Google Protobuf message or an embedded message by calling the appropriate field codecs for every field.

`#include <goby/acomms/dccl/dccl_field_codec_default_message.h>`

Inheritance diagram for goby::acomms::DCCLDefaultMessageCodec:



Additional Inherited Members

17.9.1 Detailed Description

Provides the default codec for encoding a base Google Protobuf message or an embedded message by calling the appropriate field codecs for every field.

Definition at line 38 of file dccl_field_codec_default_message.h.

The documentation for this class was generated from the following files:

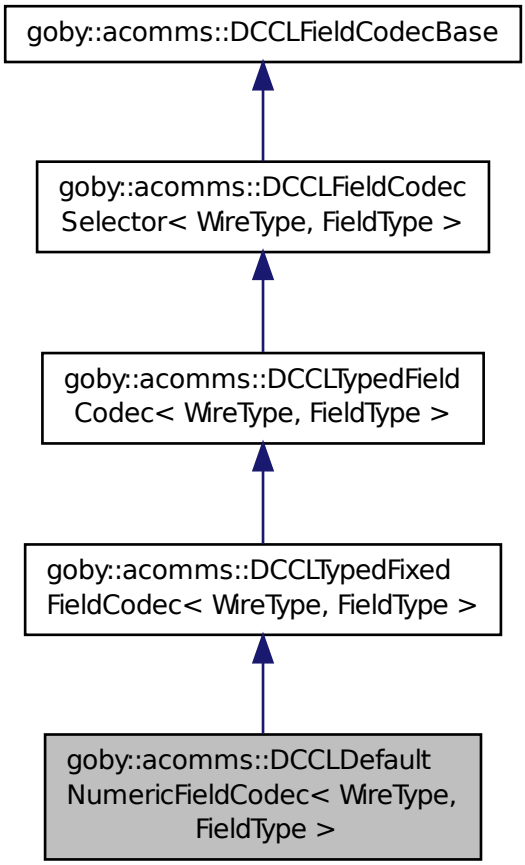
- goby/acomms/dccl/dccl_field_codec_default_message.h
- src/acomms/dccl/dccl_field_codec_default_message.cpp

17.10 goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType > Class Template Reference

Provides a basic bounded arbitrary length numeric (double, float, uint32, uint64, int32, int64) encoder.

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >:



Protected Member Functions

- virtual double **max** ()
- virtual double **min** ()
- virtual double **precision** ()
- virtual void validate ()

  Validate a field. Use require() inside your overloaded validate() to assert requirements or throw DCCLExceptions directly as needed.
- Bitset encode ()

  Encode an empty field.
- virtual Bitset encode (const WireType &value)

  Encode a non-empty field.
- virtual WireType decode (Bitset *bits)

Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValueException to indicate this.

- unsigned size ()

  The size of the encoded message in bits. Use DCCLTypedFieldCodec if the size depends on the data.

Additional Inherited Members

17.10.1 Detailed Description

template<typename WireType, typename FieldType = WireType>class goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >

Provides a basic bounded arbitrary length numeric (double, float, uint32, uint64, int32, int64) encoder.

Takes ceil(log2((max-min)∗10^precision)+1) bits for required fields, ceil(log2((max-min)∗10^precision)+2) for optional fields.

Definition at line 84 of file dccl_field_codec_default.h.

17.10.2 Member Function Documentation

17.10.2.1 template<typename WireType, typename FieldType = WireType> virtual WireType goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >::decode (Bitset ∗ *bits*) [inline], [protected], [virtual]

Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValueException to indicate this.

Parameters

bits	Bits to use for decoding.

Returns

the decoded value.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 139 of file dccl_field_codec_default.h.

17.10.2.2 template<typename WireType, typename FieldType = WireType> Bitset goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >::encode ( ) [inline], [protected], [virtual]

Encode an empty field.

Returns

Bits represented the encoded field.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 112 of file dccl_field_codec_default.h.

17.10.2.3 template<typename WireType, typename FieldType = WireType> virtual Bitset goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >::encode (const WireType & *wire_value*) [inline], [protected], [virtual]

Encode a non-empty field.

Parameters

wire_value	Value to encode.

Returns

Bits represented the encoded field.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 118 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following file:

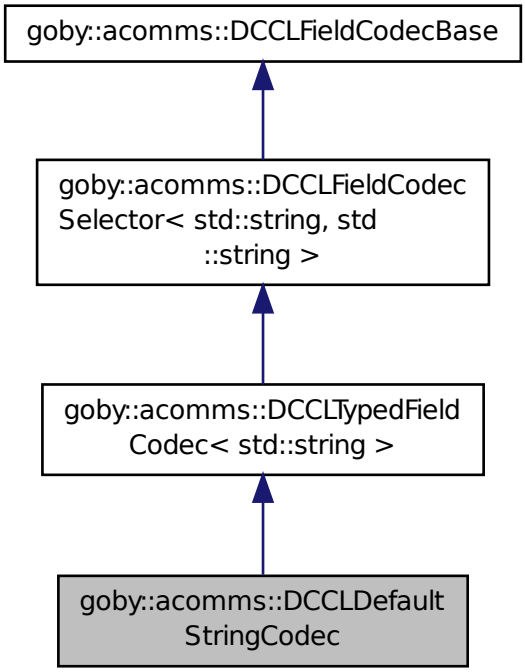
- goby/acomms/dccl/dccl_field_codec_default.h

17.11 goby::acomms::DCCLDefaultStringCodec Class Reference

Provides an variable length ASCII string encoder. Can encode strings up to 255 bytes by using a length byte preceeding the string.

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLDefaultStringCodec:



Additional Inherited Members

17.11.1 Detailed Description

Provides an variable length ASCII string encoder. Can encode strings up to 255 bytes by using a length byte preceeding the string.

[length of following string (1 byte)][string (0-255 bytes)]

Definition at line 184 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following files:

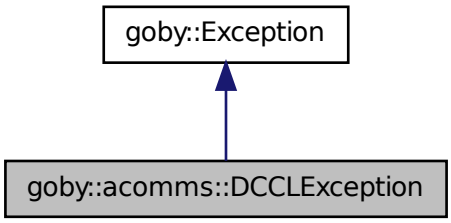
- goby/acomms/dccl/dccl_field_codec_default.h
- src/acomms/dccl/dccl_field_codec_default.cpp

17.12 goby::acomms::DCCLException Class Reference

Exception class for libdccl.

`#include <goby/acomms/dccl/dccl_exception.h>`

Inheritance diagram for goby::acomms::DCCLException:



Public Member Functions

- **DCCLException** (const std::string &s)

17.12.1 Detailed Description

Exception class for libdccl.

Definition at line 35 of file dccl_exception.h.

The documentation for this class was generated from the following file:

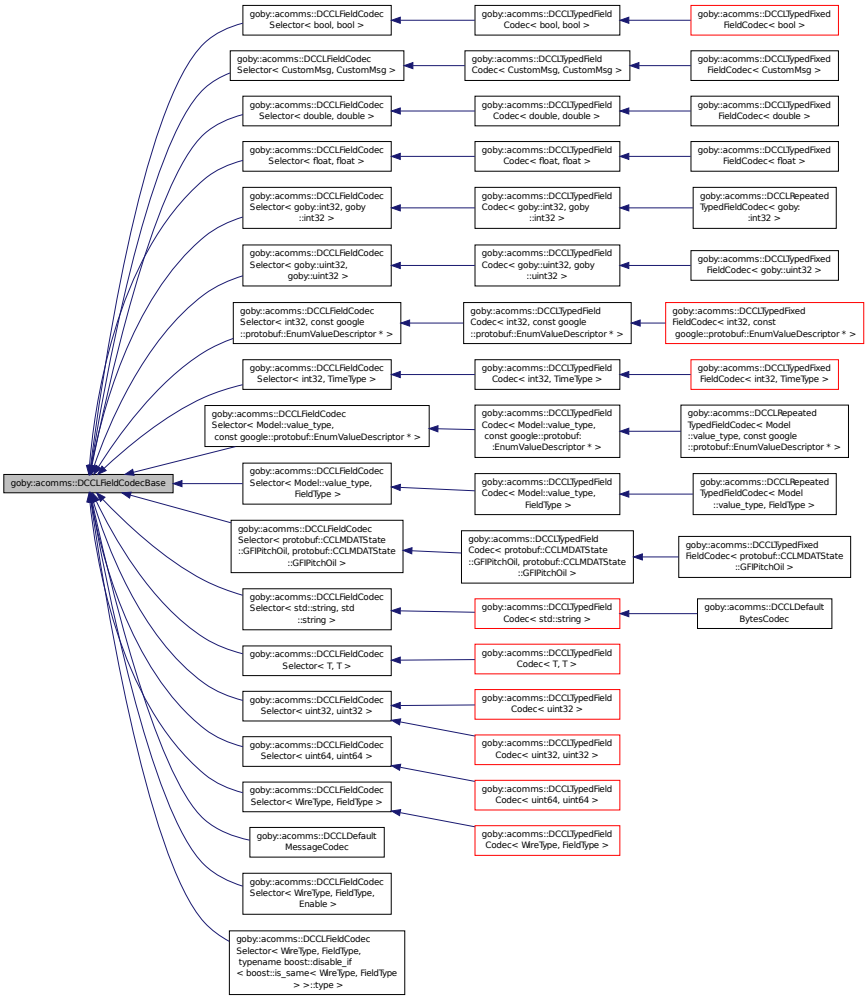
- goby/acomms/dccl/dccl_exception.h

17.13 goby::acomms::DCCLFieldCodecBase Class Reference

Provides a base class for defining DCCL field encoders / decoders. Most users will use the DCCLTypedFieldCodec or its children (e.g. DCCLTypedFixedFieldCodec) instead of directly inheriting from this class.

`#include <goby/acomms/dccl/dccl_field_codec.h>`

Inheritance diagram for goby::acomms::DCCLFieldCodecBase:



Public Member Functions

Constructor, Destructor

- **DCCLFieldCodecBase** ()
- virtual ~**DCCLFieldCodecBase** ()

Base message functions

```
These are called typically by DCCLCodec to start processing a new message. In this example "Foo"
@code
```

message Foo { int32 bar = 1; FooBar baz = 2; }

- void base_encode (Bitset *bits, const google::protobuf::Message &msg, MessageHandler::MessagePart part)

  Encode this part (body or head) of the base message.
- void base_size (unsigned *bit_size, const google::protobuf::Message &msg, MessageHandler::MessagePart part)

  Calculate the size (in bits) of a part of the base message when it is encoded.
- void base_decode (Bitset *bits, google::protobuf::Message *msg, MessageHandler::MessagePart part)

Decode part of a message.

- void base_max_size (unsigned *bit_size, const google::protobuf::Descriptor *desc, MessageHandler::MessagePart part)

  Calculate the maximum size of a message given its Descriptor alone (no data)
- void base_min_size (unsigned *bit_size, const google::protobuf::Descriptor *desc, MessageHandler::MessagePart part)

  Calculate the minimum size of a message given its Descriptor alone (no data)
- void base_validate (const google::protobuf::Descriptor *desc, MessageHandler::MessagePart part)

  Validate this part of the message to make sure all required extensions are set.
- void base_info (std::ostream *os, const google::protobuf::Descriptor *desc, MessageHandler::MessagePart part)

  Get human readable information (size of fields, etc.) about this part of the DCCL message.

Field functions (primitive types and embedded messages)

- void field_pre_encode (boost::any *wire_value, const boost::any &field_value)

  Pre-encodes a non-repeated (i.e. optional or required) field by converting the FieldType representation (the Google Protobuf representation) into the WireType representation (the type used in the encoded DCCL message). This allows for type-converting codecs.
- void field_pre_encode_repeated (std::vector< boost::any > *wire_values, const std::vector< boost::any > &field_values)

  Pre-encodes a repeated field.
- void field_encode (Bitset *bits, const boost::any &field_value, const google::protobuf::FieldDescriptor *field)

  Encode a non-repeated field.
- void field_encode_repeated (Bitset *bits, const std::vector< boost::any > &field_values, const google::protobuf::FieldDescriptor *field)

  Encode a repeated field.
- void field_size (unsigned *bit_size, const boost::any &field_value, const google::protobuf::FieldDescriptor *field)

  Calculate the size of a field.
- void field_size_repeated (unsigned *bit_size, const std::vector< boost::any > &wire_values, const google::protobuf::FieldDescriptor *field)

  Calculate the size of a repeated field.
- void field_decode (Bitset *bits, boost::any *field_value, const google::protobuf::FieldDescriptor *field)

  Decode a non-repeated field.
- void field_decode_repeated (Bitset *bits, std::vector< boost::any > *field_values, const google::protobuf::FieldDescriptor *field)

  Decode a repeated field.
- void field_post_decode (const boost::any &wire_value, boost::any *field_value)

  Post-decodes a non-repeated (i.e. optional or required) field by converting the WireType (the type used in the encoded DCCL message) representation into the FieldType representation (the Google Protobuf representation). This allows for type-converting codecs.
- void field_post_decode_repeated (const std::vector< boost::any > &wire_values, std::vector< boost::any > *field_values)

  Post-decodes a repeated field.
- void field_max_size (unsigned *bit_size, const google::protobuf::FieldDescriptor *field)

  Calculate the upper bound on this field's size (in bits)
- void field_min_size (unsigned *bit_size, const google::protobuf::FieldDescriptor *field)

  Calculate the lower bound on this field's size (in bits)
- void field_validate (bool *b, const google::protobuf::FieldDescriptor *field)

  Validate this field, checking that all required option extensions are set (e.g. (goby.field).dccl.max and (goby.field).dccl.min for arithmetic codecs)
- void field_info (std::ostream *os, const google::protobuf::FieldDescriptor *field)

  Write human readable information about the field and its bounds to the provided stream.

Protected Member Functions

- dccl::DCCLFieldOptions dccl_field_options () const

  Get the DCCL field option extension value for the current field.
- void require (bool b, const std::string &description)

  Essentially an assertion to be used in the validate() virtual method.
- virtual void any_encode (Bitset *bits, const boost::any &wire_value)=0

  Virtual method used to encode.
- virtual void any_decode (Bitset *bits, boost::any *wire_value)=0

  Virtual method used to decode.
- virtual void any_pre_encode (boost::any *wire_value, const boost::any &field_value)

  Virtual method used to pre-encode (convert from FieldType to WireType). The default implementation of this method is for when WireType == FieldType and simply copies the field_value to the wire_value.
- virtual void any_post_decode (const boost::any &wire_value, boost::any *field_value)

  Virtual method used to post-decode (convert from WireType to FieldType). The default implementation of this method is for when WireType == FieldType and simply copies the wire_value to the field_value.
- virtual unsigned any_size (const boost::any &wire_value)=0

  Virtual method for calculating the size of a field (in bits).
- virtual void validate ()

  Validate a field. Use require() inside your overloaded validate() to assert requirements or throw DCCLExceptions directly as needed.
- virtual std::string info ()

  Write field specific information (in addition to general information such as sizes that are automatically written by this class for all fields.
- virtual unsigned max_size ()=0

  Calculate maximum size of the field in bits.
- virtual unsigned min_size ()=0

  Calculate minimum size of the field in bits.
- virtual void **any_encode_repeated** (Bitset *bits, const std::vector< boost::any > &wire_values)
- virtual void **any_decode_repeated** (Bitset *repeated_bits, std::vector< boost::any > *field_values)
- virtual void **any_pre_encode_repeated** (std::vector< boost::any > *wire_values, const std::vector< boost::any > &field_values)
- virtual void **any_post_decode_repeated** (const std::vector< boost::any > &wire_values, std::vector< boost::any > *field_values)
- virtual unsigned **any_size_repeated** (const std::vector< boost::any > &wire_values)
- virtual unsigned **max_size_repeated** ()
- virtual unsigned **min_size_repeated** ()

Friends

- class **DCCLFieldCodecManager**

Information Methods

- std::string name () const

  the name of the codec used to identifier it in the .proto custom option extension
- google::protobuf::FieldDescriptor::Type field_type () const

  the type exposed to the user in the original and decoded Protobuf messages
- google::protobuf::FieldDescriptor::CppType wire_type () const

  the C++ type used "on the wire". This is the type visible after pre_encode and before post_decode functions are called.
- const google::protobuf::FieldDescriptor * this_field () const

Returns the FieldDescriptor (field schema meta-data) for this field.

- static const google::protobuf::Descriptor * this_descriptor ()

  Returns the Descriptor (message schema meta-data) for the immediate parent Message.

- static const google::protobuf::Message * **root_message** ()
- static MessageHandler::MessagePart part ()

  the part of the message currently being encoded (head or body).

17.13.1 Detailed Description

Provides a base class for defining DCCL field encoders / decoders. Most users will use the DCCLTypedFieldCodec or its children (e.g. DCCLTypedFixedFieldCodec) instead of directly inheriting from this class.

Definition at line 55 of file dccl_field_codec.h.

17.13.2 Member Function Documentation

17.13.2.1 virtual void goby::acomms::DCCLFieldCodecBase::any_decode (Bitset * *bits*, boost::any * *wire_value*) [protected], [pure virtual]

Virtual method used to decode.

Parameters

bits	Bitset containing bits to decode. This will initially contain min_size() bits. If you need more bits, call get_more_bits() with the number of bits required. This bits will be consumed from the bit pool and placed in `bits`.
wire_value	Place to store decoded value (as FieldType)

17.13.2.2 virtual void goby::acomms::DCCLFieldCodecBase::any_encode (Bitset * *bits*, const boost::any & *wire_value*) [protected], [pure virtual]

Virtual method used to encode.

Parameters

bits	Bitset to store encoded bits. Bits is *just* the bits from the current operation (unlike base_encode() and field_encode() where bits are added to the most significant end).
wire_value	Value to encode (WireType)

17.13.2.3 virtual void goby::acomms::DCCLFieldCodecBase::any_post_decode (const boost::any & *wire_value*, boost::any * *field_value*) [inline], [protected], [virtual]

Virtual method used to post-decode (convert from WireType to FieldType). The default implementation of this method is for when WireType == FieldType and simply copies the wire_value to the field_value.

Parameters

wire_value	Value to convert (WireType)
field_value	Converted value (FieldType)

Definition at line 364 of file dccl_field_codec.h.

17.13.2.4 virtual void goby::acomms::DCCLFieldCodecBase::any_pre_encode (boost::any * *wire_value*, const boost::any & *field_value*) [inline],[protected],[virtual]

Virtual method used to pre-encode (convert from FieldType to WireType). The default implementation of this method is for when WireType == FieldType and simply copies the field_value to the wire_value.

Parameters

wire_value	Converted value (WireType)
field_value	Value to convert (FieldType)

Definition at line 356 of file dccl_field_codec.h.

17.13.2.5 virtual unsigned goby::acomms::DCCLFieldCodecBase::any_size (const boost::any & *wire_value*) [protected],[pure virtual]

Virtual method for calculating the size of a field (in bits).

Parameters

field_value	Value to calculate size of

Returns

Size of field (in bits)

17.13.2.6 void goby::acomms::DCCLFieldCodecBase::base_decode (Bitset * *bits*, google::protobuf::Message * *msg*, MessageHandler::MessagePart *part*)

Decode part of a message.

Parameters

bits	Pointer to a Bitset containing bits to decode. The least significant bits will be consumed first. Any bits not consumed will remain in `bits` after this method returns.
msg	DCCL Message to *merge* the decoded result into.
part	part of the Message to decode

Definition at line 130 of file dccl_field_codec.cpp.

17.13.2.7 void goby::acomms::DCCLFieldCodecBase::base_encode (Bitset * *bits*, const google::protobuf::Message & *msg*, MessageHandler::MessagePart *part*)

Encode this part (body or head) of the base message.

Parameters

bits	pointer to a Bitset where all bits will be pushed on to the most significant end.
msg	DCCL Message to encode
part	Part of the message to encode

Definition at line 45 of file dccl_field_codec.cpp.

17.13.2.8 void goby::acomms::DCCLFieldCodecBase::base_info (std::ostream * *os*, const google::protobuf::Descriptor * *desc*, MessageHandler::MessagePart *part*)

Get human readable information (size of fields, etc.) about this part of the DCCL message.

Parameters

os	Pointer to stream to store this information
desc	Descriptor to get information on. Use google::protobuf::Message::GetDescriptor() or MyProtobufType::descriptor() to get this object.
part	the part of the Message to act on.

Definition at line 289 of file dccl_field_codec.cpp.

17.13.2.9 void goby::acomms::DCCLFieldCodecBase::base_max_size (unsigned * *bit_size,* const google::protobuf::Descriptor * *desc,* MessageHandler::MessagePart *part*)

Calculate the maximum size of a message given its Descriptor alone (no data)

Parameters

bit_size	Pointer to unsigned integer to store calculated maximum size in bits.
desc	Descriptor to calculate the maximum size of. Use google::protobuf::Message::GetDescriptor() or MyProtobufType::descriptor() to get this object.
part	part of the Message

Definition at line 202 of file dccl_field_codec.cpp.

17.13.2.10 void goby::acomms::DCCLFieldCodecBase::base_min_size (unsigned * *bit_size,* const google::protobuf::Descriptor * *desc,* MessageHandler::MessagePart *part*)

Calculate the minimum size of a message given its Descriptor alone (no data)

Parameters

bit_size	Pointer to unsigned integer to store calculated minimum size in bits.
desc	Descriptor to calculate the minimum size of. Use google::protobuf::Message::GetDescriptor() or MyProtobufType::descriptor() to get this object.
part	part of the Message

Definition at line 232 of file dccl_field_codec.cpp.

17.13.2.11 void goby::acomms::DCCLFieldCodecBase::base_size (unsigned * *bit_size,* const google::protobuf::Message & *msg,* MessageHandler::MessagePart *part*)

Calculate the size (in bits) of a part of the base message when it is encoded.

Parameters

bit_size	Pointer to unsigned integer to store the result.
msg	the DCCL Message of which to calculate the size
part	part of the Message to calculate the size of

Definition at line 91 of file dccl_field_codec.cpp.

17.13.2.12 void goby::acomms::DCCLFieldCodecBase::base_validate (const google::protobuf::Descriptor * *desc,* MessageHandler::MessagePart *part*)

Validate this part of the message to make sure all required extensions are set.

Parameters

desc	Descriptor to validate. Use google::protobuf::Message::GetDescriptor() or MyProtobufType::descriptor() to get this object.
part	part of the Message

Definition at line 262 of file dccl_field_codec.cpp.

17.13.2.13 dccl::DCCLFieldOptions goby::acomms::DCCLFieldCodecBase::dccl_field_options ( ) const [inline], [protected]

Get the DCCL field option extension value for the current field.

dccl::DCCLFieldOptions is defined in acomms_option_extensions.proto

Definition at line 309 of file dccl_field_codec.h.

17.13.2.14 void goby::acomms::DCCLFieldCodecBase::field_decode (Bitset * *bits*, boost::any * *field_value*, const google::protobuf::FieldDescriptor * *field*)

Decode a non-repeated field.

Parameters

bits	Bits to decode. Used bits are consumed (erased) from the least significant end
field_value	Location to store decoded value (FieldType)
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 141 of file dccl_field_codec.cpp.

17.13.2.15 void goby::acomms::DCCLFieldCodecBase::field_decode_repeated (Bitset * *bits*, std::vector< boost::any > * *field_values*, const google::protobuf::FieldDescriptor * *field*)

Decode a repeated field.

Parameters

bits	Bits to decode. Used bits are consumed (erased) from the least significant end
field_values	Location to store decoded values (FieldType)
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 172 of file dccl_field_codec.cpp.

17.13.2.16 void goby::acomms::DCCLFieldCodecBase::field_encode (Bitset * *bits*, const boost::any & *field_value*, const google::protobuf::FieldDescriptor * *field*)

Encode a non-repeated field.

Parameters

bits	Pointer to bitset to store encoded bits. Bits are added to the most significant end of `bits`
field_value	Value to encode (FieldType)
field	Protobuf descriptor to the field to encode. Set to 0 for base message.

Definition at line 59 of file dccl_field_codec.cpp.

17.13.2.17 void goby::acomms::DCCLFieldCodecBase::field_encode_repeated (Bitset * *bits*, const std::vector< boost::any > & *field_values*, const google::protobuf::FieldDescriptor * *field*)

Encode a repeated field.

Parameters

bits	Pointer to bitset to store encoded bits. Bits are added to the most significant end of `bits`
field_values	Values to encode (FieldType)
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 76 of file dccl_field_codec.cpp.

17.13.2.18 void goby::acomms::DCCLFieldCodecBase::field_info (std::ostream * *os,* const google::protobuf::FieldDescriptor * *field*)

Write human readable information about the field and its bounds to the provided stream.

Parameters

os	Stream to write info to.
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 303 of file dccl_field_codec.cpp.

17.13.2.19 void goby::acomms::DCCLFieldCodecBase::field_max_size (unsigned * *bit_size,* const google::protobuf::FieldDescriptor * *field*)

Calculate the upper bound on this field's size (in bits)

Parameters

bit_size	Location to *add* calculated maximum bit size to. Be sure to zero `bit_size` if you want only the size of this field.
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 219 of file dccl_field_codec.cpp.

17.13.2.20 void goby::acomms::DCCLFieldCodecBase::field_min_size (unsigned * *bit_size,* const google::protobuf::FieldDescriptor * *field*)

Calculate the lower bound on this field's size (in bits)

Parameters

bit_size	Location to *add* calculated minimum bit size to. Be sure to zero `bit_size` if you want only the size of this field.
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 249 of file dccl_field_codec.cpp.

17.13.2.21 void goby::acomms::DCCLFieldCodecBase::field_post_decode (const boost::any & *wire_value,* boost::any * *field_value*) [inline]

Post-decodes a non-repeated (i.e. optional or required) field by converting the WireType (the type used in the encoded DCCL message) representation into the FieldType representation (the Google Protobuf representation). This allows for type-converting codecs.

Parameters

wire_value	Should be set to the desired value to translate
field_value	Will be set to the converted wire_value

Definition at line 264 of file dccl_field_codec.h.

17.13.2.22 void goby::acomms::DCCLFieldCodecBase::field_post_decode_repeated (const std::vector< boost::any > & *wire_values,* std::vector< boost::any > * *field_values*) [inline]

Post-decodes a repeated field.

Parameters

wire_values	Should be set to the desired values to translate
field_values	Will be set to the converted wire_values

Definition at line 271 of file dccl_field_codec.h.

17.13.2.23 void goby::acomms::DCCLFieldCodecBase::field_pre_encode (boost::any * *wire_value,* const boost::any & *field_value*) [inline]

Pre-encodes a non-repeated (i.e. optional or required) field by converting the FieldType representation (the Google Protobuf representation) into the WireType representation (the type used in the encoded DCCL message). This allows for type-converting codecs.

```
These are called typically by DCCLDefaultMessageCodec to start processing a new field. In this exam
@code
```

message Foo { int32 bar = 1; FooBar baz = 2; }

Parameters

wire_value	Will be set to the converted field_value
field_value	Value to convert to the appropriate wire_value

Definition at line 193 of file dccl_field_codec.h.

17.13.2.24 void goby::acomms::DCCLFieldCodecBase::field_pre_encode_repeated (std::vector< boost::any > * *wire_values,* const std::vector< boost::any > & *field_values*) [inline]

Pre-encodes a repeated field.

Parameters

wire_values	Should be set to the converted field_values
field_values	Values to convert to the appropriate wire_values

Definition at line 200 of file dccl_field_codec.h.

17.13.2.25 void goby::acomms::DCCLFieldCodecBase::field_size (unsigned * *bit_size,* const boost::any & *field_value,* const google::protobuf::FieldDescriptor * *field*)

Calculate the size of a field.

Parameters

bit_size	Location to *add* calculated bit size to. Be sure to zero `bit_size` if you want only the size of this field.
field_value	Value calculate size of (FieldType)
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 102 of file dccl_field_codec.cpp.

17.13.2.26 void goby::acomms::DCCLFieldCodecBase::field_size_repeated (unsigned * *bit_size,* const std::vector< boost::any > & *wire_values,* const google::protobuf::FieldDescriptor * *field*)

Calculate the size of a repeated field.

Parameters

bit_size	Location to *add* calculated bit size to. Be sure to zero `bit_size` if you want only the size of this field.

field_values	Values to calculate size of (FieldType)
field	Protobuf descriptor to the field. Set to 0 for base message.

Definition at line 115 of file dccl_field_codec.cpp.

17.13.2.27 google::protobuf::FieldDescriptor::Type goby::acomms::DCCLFieldCodecBase::field_type ( ) const [inline]

the type exposed to the user in the original and decoded Protobuf messages

Returns

the Google Protobuf message type. See http://code.google.com/apis/protocolbuffers/docs/reference/protobuf.descriptor.html#FieldDescriptor.Type.details

Definition at line 72 of file dccl_field_codec.h.

17.13.2.28 void goby::acomms::DCCLFieldCodecBase::field_validate (bool * *b*, const google::protobuf::FieldDescriptor * *field*)

Validate this field, checking that all required option extensions are set (e.g. (goby.field).dccl.max and (goby.field).dccl.min for arithmetic codecs)

Parameters

b	Currently unused (will be set to false)
field	Protobuf descriptor to the field. Set to 0 for base message.

Exceptions

DCCLException	If field is invalid

Definition at line 278 of file dccl_field_codec.cpp.

17.13.2.29 std::string goby::acomms::DCCLFieldCodecBase::info ( ) [protected], [virtual]

Write field specific information (in addition to general information such as sizes that are automatically written by this class for all fields.

Returns

string containing information to display.

Definition at line 365 of file dccl_field_codec.cpp.

17.13.2.30 virtual unsigned goby::acomms::DCCLFieldCodecBase::max_size ( ) [protected], [pure virtual]

Calculate maximum size of the field in bits.

Returns

Maximum size of this field (in bits).

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, and goby::acomms::DCCLDefaultIdentifierCodec.

17.13.2.31 virtual unsigned goby::acomms::DCCLFieldCodecBase::min_size ( ) [protected], [pure virtual]

Calculate minimum size of the field in bits.

Returns

Minimum size of this field (in bits).

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, and goby::acomms::DCCLDefaultIdentifierCodec.

17.13.2.32 void goby::acomms::DCCLFieldCodecBase::require (bool *b*, const std::string & *description*) [inline], [protected]

Essentially an assertion to be used in the validate() virtual method.

Parameters

b	Boolean to assert (if true, execution continues, if false an exception is thrown)
description	Debugging description for this assertion (will be appended to the exception)

Exceptions

DCCLException	Thrown if !b

Definition at line 323 of file dccl_field_codec.h.

17.13.2.33 static const google::protobuf::Descriptor* goby::acomms::DCCLFieldCodecBase::this_descriptor ( ) [inline], [static]

Returns the Descriptor (message schema meta-data) for the immediate parent Message.

```
for:
@code
```

message Foo { int32 bar = 1; FooBar baz = 2; } returns Descriptor for Foo if this_field() == 0 returns Descriptor for Foo if this_field() == FieldDescriptor for bar returns Descriptor for FooBar if this_field() == FieldDescriptor for baz

Definition at line 99 of file dccl_field_codec.h.

17.13.2.34 const google::protobuf::FieldDescriptor* goby::acomms::DCCLFieldCodecBase::this_field ( ) const [inline]

Returns the FieldDescriptor (field schema meta-data) for this field.

Returns

FieldDescriptor for the current field or 0 if this codec is encoding the base message.

Definition at line 83 of file dccl_field_codec.h.

17.13.2.35 google::protobuf::FieldDescriptor::CppType goby::acomms::DCCLFieldCodecBase::wire_type ( ) const [inline]

the C++ type used "on the wire". This is the type visible *after* pre_encode and *before* post_decode functions are called.

The wire type allows codecs to make type changes (e.g. from string to integer) before reusing another codec that knows how to encode that wire type (e.g. DCCLDefaultNumericFieldCodec)

Returns

the C++ type used to encode and decode. See http://code.google.com/apis/protocolbuffers/docs/refer
protobuf.descriptor.html#FieldDescriptor.CppType.details

Definition at line 77 of file dccl_field_codec.h.

The documentation for this class was generated from the following files:

- goby/acomms/dccl/dccl_field_codec.h
- src/acomms/dccl/dccl_field_codec.cpp

17.14 goby::acomms::DCCLFieldCodecManager Class Reference

`#include <goby/acomms/dccl/dccl_field_codec_manager.h>`

Static Public Member Functions

- template<class Codec >
  static boost::enable_if
  < boost::mpl::and_
  < boost::is_base_of
  < google::protobuf::Message,
  typename Codec::wire_type >
  , boost::mpl::not_
  < boost::is_same
  < google::protobuf::Message,
  typename Codec::wire_type >
  > >, void >::type add (const std::string &name)

  Add a new field codec (used for codecs operating on statically generated Protobuf messages, that is, children of google::protobuf::Message but not google::protobuf::Message itself).
- template<class Codec >
  static boost::disable_if
  < boost::mpl::and_
  < boost::is_base_of
  < google::protobuf::Message,
  typename Codec::wire_type >
  , boost::mpl::not_
  < boost::is_same
  < google::protobuf::Message,
  typename Codec::wire_type >
  > >, void >::type add (const std::string &name)

  Add a new field codec (used for codecs operating on all types except statically generated Protobuf messages).
- template<class Codec , google::protobuf::FieldDescriptor::Type type>
  static void add (const std::string &name)

  Add a new field codec only valid for a specific google::protobuf::FieldDescriptor::Type. This is useful if a given codec is designed to work with only a specific Protobuf type that shares an underlying C++ type (e.g. Protobuf types `bytes` and `string`)
- static boost::shared_ptr
  < DCCLFieldCodecBase > find (const google::protobuf::FieldDescriptor *field)

  Find the codec for a given field. For embedded messages, prefers (goby.field).dccl.codec (inside field) over (goby.msg).dccl.codec (inside embedded message).
- static boost::shared_ptr
  < DCCLFieldCodecBase > find (const google::protobuf::Descriptor *desc, std::string name="")

  Find the codec for a given base (or embedded) message.

17.14.1 Detailed Description

Todo (tes) Make sanity check for newly added FieldCodecs

Definition at line 37 of file dccl_field_codec_manager.h.

17.14.2 Member Function Documentation

17.14.2.1 template<class Codec , google::protobuf::FieldDescriptor::Type type> void goby::acomms::DCCLFieldCodecManager::add (const std::string & *name*) [static]

Add a new field codec (used for codecs operating on statically generated Protobuf messages, that is, children of google::protobuf::Message but not google::protobuf::Message itself).

Template Parameters

Codec	A child of DCCLFieldCodecBase

Parameters

name	Name to use for this codec. Corresponds to (goby.field).dccl.codec="name" in .proto file.

Returns

nothing (void). Return templates are used for template metaprogramming selection of the proper add() overload.

Definition at line 159 of file dccl_field_codec_manager.h.

17.14.2.2 template<class Codec > static boost::disable_if< boost::mpl::and_<boost::is_base_of<google::protobuf::Message, typename Codec::wire_type>, boost::mpl::not_<boost::is_same<google::protobuf::Message,typename Codec::wire_type> > >, void>::type goby::acomms::DCCLFieldCodecManager::add (const std::string & *name*) [static]

Add a new field codec (used for codecs operating on all types except statically generated Protobuf messages).

Template Parameters

Codec	A child of DCCLFieldCodecBase

Parameters

name	Name to use for this codec. Corresponds to (goby.field).dccl.codec="name" in .proto file.

Returns

nothing (void). Return templates are used for template metaprogramming selection of the proper add() overload.

17.14.2.3 template<class Codec , google::protobuf::FieldDescriptor::Type type> static void goby::acomms::DCCLFieldCodecManager::add (const std::string & *name*) [static]

Add a new field codec only valid for a specific google::protobuf::FieldDescriptor::Type. This is useful if a given codec is designed to work with only a specific Protobuf type that shares an underlying C++ type (e.g. Protobuf types `bytes` and `string`)

Template Parameters

Codec	A child of DCCLFieldCodecBase
type	The google::protobuf::FieldDescriptor::Type enumeration that this codec works on.

Parameters

name	Name to use for this codec. Corresponds to (goby.field).dccl.codec="name" in .proto file.

17.14.2.4 static boost::shared_ptr<DCCLFieldCodecBase> goby::acomms::DCCLFieldCodecManager::find (const google::protobuf::Descriptor * *desc*, std::string *name* = " ") [inline], [static]

Find the codec for a given base (or embedded) message.

Parameters

desc	Message descriptor to find codec for
name	Codec name (used for embedded messages to prefer the codec listed as a field option). Omit for finding the codec of a base message (one that is not embedded).

Definition at line 94 of file dccl_field_codec_manager.h.

The documentation for this class was generated from the following files:

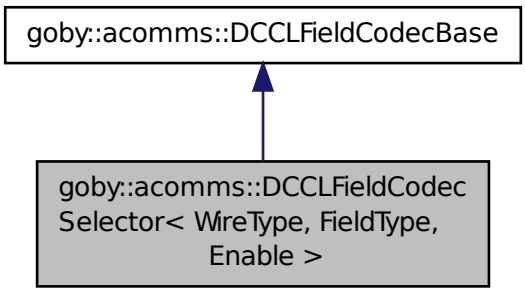
- goby/acomms/dccl/dccl_field_codec_manager.h
- src/acomms/dccl/dccl_field_codec_manager.cpp

17.15 goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable > Class Template Reference

if WireType == FieldType, we don't have to add any more virtual methods for converting between them.

`#include <goby/acomms/dccl/dccl_field_codec_typed.h>`

Inheritance diagram for goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable >:



Additional Inherited Members

17.15.1 Detailed Description

template<typename WireType, typename FieldType, class Enable = void>class goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable >

if WireType == FieldType, we don't have to add any more virtual methods for converting between them.

Definition at line 37 of file dccl_field_codec_typed.h.

The documentation for this class was generated from the following file:

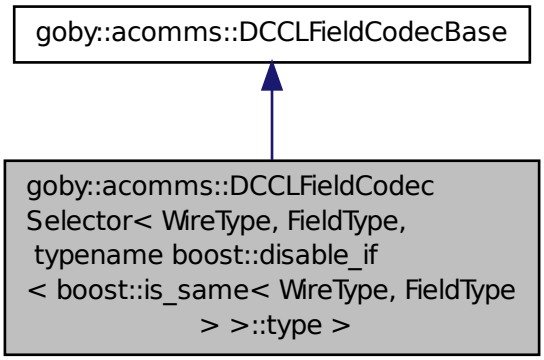
- goby/acomms/dccl/dccl_field_codec_typed.h

17.16 goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type > Class Template Reference

If WireType != FieldType, adds some more pure virtual methods to handle the type conversions (pre_encode() and post_decode()). If WireType == FieldType this class is not inherited and this pure virtual methods do not exist (and thus can be omitted in the child class).

`#include <goby/acomms/dccl/dccl_field_codec_typed.h>`

Inheritance diagram for goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type >:



Protected Member Functions

- virtual WireType pre_encode (const FieldType &field_value)=0

  Convert from the FieldType representation (used in the Google Protobuf message) to the WireType representation (used with encode() and decode(), i.e. "on the wire").
- virtual FieldType post_decode (const WireType &wire_value)=0

  Convert from the WireType representation (used with encode() and decode(), i.e. "on the wire") to the FieldType representation (used in the Google Protobuf message).

Additional Inherited Members

17.16.1 Detailed Description

template<typename WireType, typename FieldType>class goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type >

If WireType != FieldType, adds some more pure virtual methods to handle the type conversions (pre_encode() and post_decode()). If WireType == FieldType this class is not inherited and this pure virtual methods do not exist (and thus can be omitted in the child class).

Definition at line 44 of file dccl_field_codec_typed.h.

17.16.2 Member Function Documentation

17.16.2.1 template<typename WireType , typename FieldType > virtual FieldType goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type >::post_decode (const WireType & *wire_value*) [protected], [pure virtual]

Convert from the WireType representation (used with encode() and decode(), i.e. "on the wire") to the FieldType representation (used in the Google Protobuf message).

Parameters

wire_value	Value to convert

Returns

Converted value

17.16.2.2 template<typename WireType , typename FieldType > virtual WireType goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if< boost::is_same< WireType, FieldType > >::type >::pre_encode (const FieldType & *field_value*) [protected], [pure virtual]

Convert from the FieldType representation (used in the Google Protobuf message) to the WireType representation (used with encode() and decode(), i.e. "on the wire").

Parameters

field_value	Value to convert

Returns

Converted value

The documentation for this class was generated from the following file:

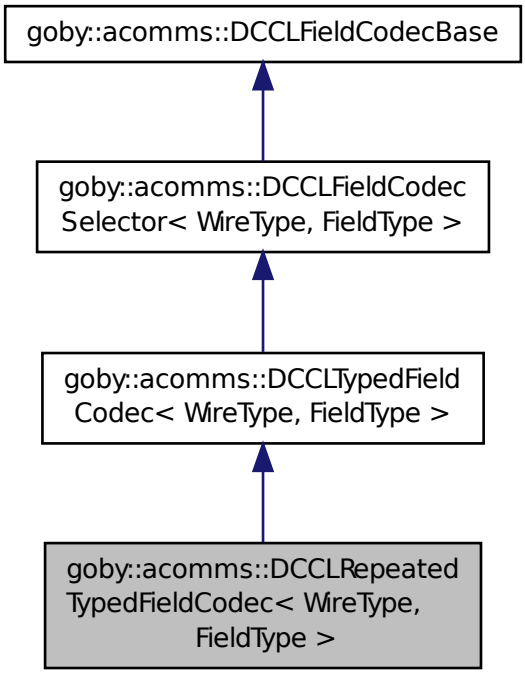
- goby/acomms/dccl/dccl_field_codec_typed.h

17.17 goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType > Class Template Reference

Base class for "repeated" (multiple value) static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire). Use DCCLTypedFieldCodec if your fields are always singular ("optional" or "required"). Singular fields are default implemented in this codec by calls to the equivalent repeated function with an empty or single valued vector.

```
#include <goby/acomms/dccl/dccl_field_codec_typed.h>
```

Inheritance diagram for goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >:



Public Types

- typedef WireType **wire_type**
- typedef FieldType **field_type**

Public Member Functions

- virtual Bitset encode_repeated (const std::vector< WireType > &wire_value)=0

  Encode a repeated field.
- virtual std::vector< WireType > decode_repeated (Bitset *bits)=0

  Decode a repeated field.
- virtual unsigned size_repeated (const std::vector< WireType > &wire_values)=0

  Give the size of a repeated field.
- virtual unsigned max_size_repeated ()=0

  Give the max size of a repeated field.
- virtual unsigned min_size_repeated ()=0

  Give the min size of a repeated field.
- virtual Bitset encode ()

  Encode an empty field.
- virtual Bitset encode (const WireType &wire_value)

  Encode a non-empty field.
- virtual WireType decode (Bitset *bits)

Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValueException to indicate this.

- virtual unsigned size ()

  Calculate the size (in bits) of an empty field.
- virtual unsigned size (const WireType &wire_value)

  Calculate the size (in bits) of a non-empty field.
- virtual unsigned max_size ()

  Calculate maximum size of the field in bits.
- virtual unsigned min_size ()

  Calculate minimum size of the field in bits.

Additional Inherited Members

17.17.1 Detailed Description

template<typename WireType, typename FieldType = WireType>class goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >

Base class for "repeated" (multiple value) static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire). Use DCCLTypedFieldCodec if your fields are always singular ("optional" or "required"). Singular fields are default implemented in this codec by calls to the equivalent repeated function with an empty or single valued vector.

Template Parameters

WireType	the type used for the encode and decode functions. This can be any C++ type, and is often the same as FieldType, unless a type conversion should be performed. The reason for using a different WireType and FieldType should be clear from the DCCLDefaultEnumCodec which uses DCCLDefaultNumericFieldCodec to do all the numerical encoding / decoding while DCCLDefaultEnumCodec does the type conversion (pre_encode() and post_decode()).
FieldType	the type used in the Google Protobuf message that is exposed to the end-user DCCLCodec::decode(), DCCLCodec::encode(), etc. functions.

Definition at line 199 of file dccl_field_codec_typed.h.

17.17.2 Member Function Documentation

17.17.2.1 template<typename WireType, typename FieldType = WireType> virtual WireType goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::decode (Bitset * *bits*) [inline], [virtual]

Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValueException to indicate this.

Parameters

bits	Bits to use for decoding.

Returns

the decoded value.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 241 of file dccl_field_codec_typed.h.

17.17.2.2 template<typename WireType, typename FieldType = WireType> virtual Bitset goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::encode ( ) [inline], [virtual]

Encode an empty field.

Returns

Bits represented the encoded field.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 227 of file dccl_field_codec_typed.h.

17.17.2.3 template<typename WireType, typename FieldType = WireType> virtual Bitset goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::encode (const WireType & *wire_value*) [inline], [virtual]

Encode a non-empty field.

Parameters

wire_value	Value to encode.

Returns

Bits represented the encoded field.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 234 of file dccl_field_codec_typed.h.

17.17.2.4 template<typename WireType, typename FieldType = WireType> virtual unsigned goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::max_size ( ) [inline], [virtual]

Calculate maximum size of the field in bits.

Returns

Maximum size of this field (in bits).

Implements goby::acomms::DCCLFieldCodecBase.

Definition at line 263 of file dccl_field_codec_typed.h.

17.17.2.5 template<typename WireType, typename FieldType = WireType> virtual unsigned goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::min_size ( ) [inline], [virtual]

Calculate minimum size of the field in bits.

Returns

Minimum size of this field (in bits).

Implements goby::acomms::DCCLFieldCodecBase.

Definition at line 266 of file dccl_field_codec_typed.h.

17.17.2.6 template<typename WireType, typename FieldType = WireType> virtual unsigned goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::size ( ) [inline], [virtual]

Calculate the size (in bits) of an empty field.

Returns

the size (in bits) of the empty field.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 253 of file dccl_field_codec_typed.h.

17.17.2.7 template<typename WireType, typename FieldType = WireType> virtual unsigned goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >::size (const WireType & *wire_value*) [inline], [virtual]

Calculate the size (in bits) of a non-empty field.

Parameters

wire_value	Value to use when calculating the size of the field. If calculating the size requires encoding the field completely, cache the encoded value for a likely future call to encode() for the same wire_value.

Returns

the size (in bits) of the field.

Implements goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >.

Definition at line 260 of file dccl_field_codec_typed.h.

The documentation for this class was generated from the following file:

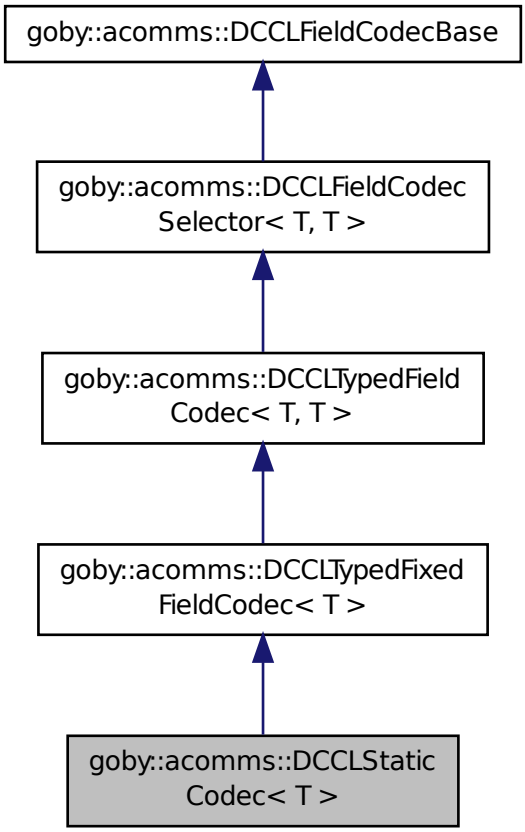
- goby/acomms/dccl/dccl_field_codec_typed.h

17.18 goby::acomms::DCCLStaticCodec< T > Class Template Reference

Placeholder codec that takes no space on the wire (0 bits).

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLStaticCodec< T >:



Additional Inherited Members

17.18.1 Detailed Description

template<typename T>class goby::acomms::DCCLStaticCodec< T >

Placeholder codec that takes no space on the wire (0 bits).

Definition at line 282 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following file:

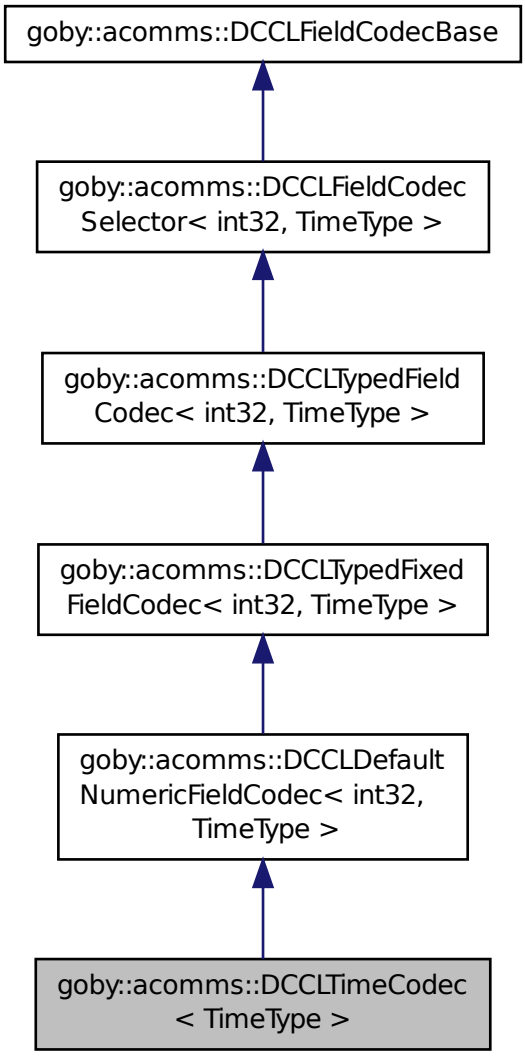
- goby/acomms/dccl/dccl_field_codec_default.h

17.19 goby::acomms::DCCLTimeCodec< TimeType > Class Template Reference

Encodes time of day (second precision) for times represented by the string representation of boost::posix_time-::ptime (e.g. obtained from goby_time<std::string>()).

`#include <goby/acomms/dccl/dccl_field_codec_default.h>`

Inheritance diagram for goby::acomms::DCCLTimeCodec< TimeType >:



Public Member Functions

- goby::int32 **pre_encode** (const TimeType &field_value)
- TimeType **post_decode** (const int32 &wire_value)

Additional Inherited Members

17.19.1 Detailed Description

template<typename TimeType>class goby::acomms::DCCLTimeCodec< TimeType >

Encodes time of day (second precision) for times represented by the string representation of boost::posix_time-::ptime (e.g. obtained from goby_time<std::string>()).

Template Parameters

TimeType	A type representing time: See the various specializations of goby_time() for allowed types.

Definition at line 240 of file dccl_field_codec_default.h.

The documentation for this class was generated from the following file:

- goby/acomms/dccl/dccl_field_codec_default.h

17.20 goby::acomms::DCCLTypedFieldCodec< WireType, FieldType > Class Template Reference

Base class for static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire).

`#include <goby/acomms/dccl/dccl_field_codec_typed.h>`

Inheritance diagram for goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >:

Public Types

- typedef WireType **wire_type**
- typedef FieldType **field_type**

Public Member Functions

- virtual Bitset encode ()=0

  Encode an empty field.
- virtual Bitset encode (const WireType &wire_value)=0

  Encode a non-empty field.
- virtual WireType decode (Bitset *bits)=0

  Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValueException to indicate this.
- virtual unsigned size ()=0

  Calculate the size (in bits) of an empty field.
- virtual unsigned size (const WireType &wire_value)=0

  Calculate the size (in bits) of a non-empty field.

Additional Inherited Members

17.20.1 Detailed Description

template<typename WireType, typename FieldType = WireType>class goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >

Base class for static-typed (no boost::any) field encoders/decoders. Most user defined variable length codecs will start with this class. Use DCCLTypedFixedFieldCodec if your codec is fixed length (always uses the same number of bits on the wire).

Template Parameters

WireType	the type used for the encode and decode functions. This can be any C++ type, and is often the same as FieldType, unless a type conversion should be performed. The reason for using a different WireType and FieldType should be clear from the DCCLDefaultEnumCodec which uses DCCLDefaultNumericFieldCodec to do all the numerical encoding / decoding while DCCLDefaultEnumCodec does the type conversion (pre_encode() and post_decode()).
FieldType	the type used in the Google Protobuf message that is exposed to the end-user DCCLCodec::decode(), DCCLCodec::encode(), etc. functions.

Definition at line 105 of file dccl_field_codec_typed.h.

17.20.2 Member Function Documentation

17.20.2.1 template<typename WireType, typename FieldType = WireType> virtual WireType goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >::decode (Bitset * *bits*) [pure virtual]

Decode a field. If the field is empty (i.e. was encoded using the zero-argument encode()), throw DCCLNullValueException to indicate this.

Parameters

bits	Bits to use for decoding.

Returns

the decoded value.

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >, goby::acomms::DCCLDefaultNumericFieldCodec< uint32 >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, TimeType >, and goby::acomms::DCCLDefaultIdentifierCodec.

17.20.2.2 template<typename WireType, typename FieldType = WireType> virtual Bitset goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >::encode ( ) [pure virtual]

Encode an empty field.

Returns

Bits represented the encoded field.

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >, goby::acomms::DCCLDefaultNumericFieldCodec< uint32 >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, TimeType >, and goby::acomms::DCCLDefaultIdentifierCodec.

17.20.2.3 template<typename WireType, typename FieldType = WireType> virtual Bitset goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >::encode (const WireType & *wire_value*) [pure virtual]

Encode a non-empty field.

Parameters

wire_value	Value to encode.

Returns

Bits represented the encoded field.

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >, goby::acomms::DCCLDefaultNumericFieldCodec< uint32 >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, TimeType >, and goby::acomms::DCCLDefaultIdentifierCodec.

17.20.2.4 template<typename WireType, typename FieldType = WireType> virtual unsigned goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >::size ( ) [pure virtual]

Calculate the size (in bits) of an empty field.

Returns

the size (in bits) of the empty field.

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >, goby::acomms::DCCLDefaultNumericFieldCodec< uint32 >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLDefaultNumericFieldCodec< int32, TimeType >, goby::acomms::DCCLDefaultIdentifierCodec, goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType >, goby::acomms::DCCLTypedFixedFieldCodec< protobuf::CCLMDATState::GFIPitchOil >, goby::acomms::DCCLTypedFixedFieldCodec< double >, goby::acomms::DCCLTypedFixedFieldCodec< uint32, uint32 >, goby::acomms::DCCLTypedFixedFieldCodec< float >, goby::acomms::DCCLTypedFixedFieldCodec< CustomMsg >, goby::acomms::DCCLTypedFixedFieldCodec< T >, goby::acomms::DCCLTypedFixedFieldCodec< uint64 >, goby::acomms::DCCLTypedFixedFieldCodec< int32, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLTypedFixedFieldCodec< bool >, goby::acomms::DCCLTypedFixedFieldCodec< int32, TimeType >, and goby::acomms::DCCLTypedFixedFieldCodec< goby::uint32 >.

17.20.2.5 template<typename WireType, typename FieldType = WireType> virtual unsigned goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >::size (const WireType & *wire_value*) [pure virtual]

Calculate the size (in bits) of a non-empty field.

Parameters

wire_value	Value to use when calculating the size of the field. If calculating the size requires encoding the field completely, cache the encoded value for a likely future call to encode() for the same wire_value.

Returns

the size (in bits) of the field.

Implemented in goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, FieldType >, goby::acomms::DCCLRepeatedTypedFieldCodec< Model::value_type, const google::protobuf::EnumValueDescriptor * >, goby::acomms::DCCLRepeatedTypedFieldCodec< goby::int32 >, and goby::acomms::DCCLDefaultIdentifierCodec.

The documentation for this class was generated from the following file:

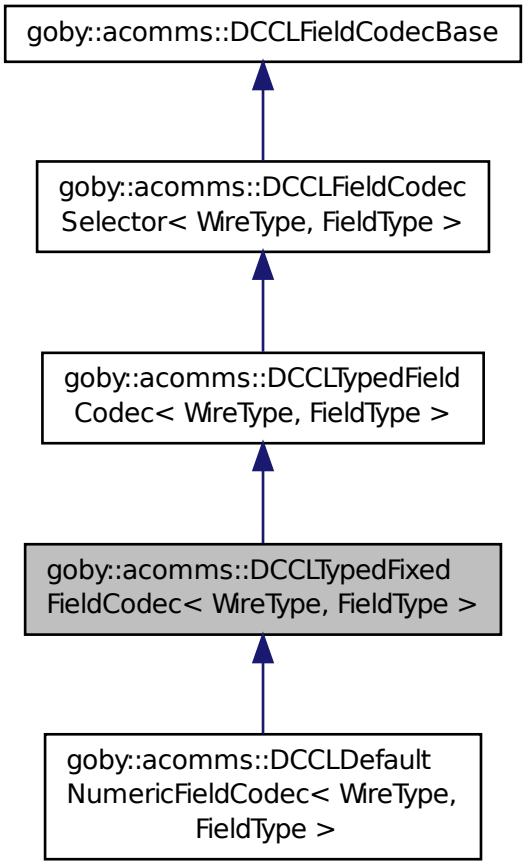
- goby/acomms/dccl/dccl_field_codec_typed.h

17.21 goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType > Class Template Reference

Base class for static-typed field encoders/decoders that use a fixed number of bits on the wire regardless of the value of the field. Use DCCLTypedFieldCodec if your encoder is variable length. See DCCLTypedFieldCodec for an explanation of the template parameters (FieldType and WireType).

`#include <goby/acomms/dccl/dccl_field_codec_fixed.h>`

Inheritance diagram for goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType >:



Protected Member Functions

- virtual unsigned size ()=0

  The size of the encoded message in bits. Use DCCLTypedFieldCodec if the size depends on the data.

Additional Inherited Members

17.21.1 Detailed Description

template<typename WireType, typename FieldType = WireType>class goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType >

Base class for static-typed field encoders/decoders that use a fixed number of bits on the wire regardless of the value of the field. Use DCCLTypedFieldCodec if your encoder is variable length. See DCCLTypedFieldCodec for an explanation of the template parameters (FieldType and WireType).

Implements DCCLTypedFieldCodec::size(const FieldType& field_value), DCCLTypedFieldCodec::max_size and DCCLTypedFieldCodec::min_size, and provides a virtual zero-argument function for size()

Definition at line 39 of file dccl_field_codec_fixed.h.

The documentation for this class was generated from the following file:

- goby/acomms/dccl/dccl_field_codec_fixed.h

17.22 goby::acomms::DCCLTypeHelper Class Reference

Provides FromProtoTypeBase and FromProtoCppTypeBase type identification helper classes for various representations of the underlying field.

`#include <goby/acomms/dccl/dccl_type_helper.h>`

Static Public Member Functions

- static boost::shared_ptr < FromProtoTypeBase > **find** (google::protobuf::FieldDescriptor::Type type)
- static boost::shared_ptr < FromProtoCppTypeBase > **find** (const google::protobuf::FieldDescriptor *field)
- static boost::shared_ptr < FromProtoCppTypeBase > **find** (const google::protobuf::Descriptor *desc)
- static boost::shared_ptr < goby::acomms::FromProtoCppTypeBase > **find** (google::protobuf::FieldDescriptor::CppType cpptype, const std::string &type_name="")

Friends

- class **DCCLFieldCodecManager**
- template<typename T > void **boost::checked_delete** (T *)

17.22.1 Detailed Description

Provides FromProtoTypeBase and FromProtoCppTypeBase type identification helper classes for various representations of the underlying field.

Definition at line 45 of file dccl_type_helper.h.

The documentation for this class was generated from the following files:

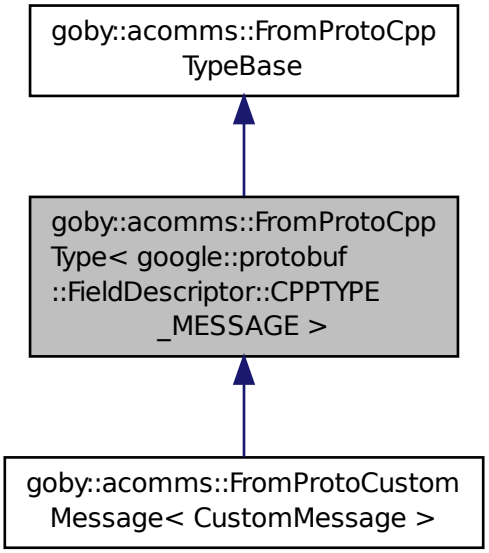
- goby/acomms/dccl/dccl_type_helper.h
- src/acomms/dccl/dccl_type_helper.cpp

17.23 goby::acomms::FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE > Class Template Reference

Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using the dynamic google::protobuf::Message as the underlying class.

`#include <goby/acomms/dccl/protobuf_cpp_type_helpers.h>`

Inheritance diagram for goby::acomms::FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE >:



Public Types

- typedef const google::protobuf::Message * **const_type**
- typedef google::protobuf::Message * **mutable_type**

Public Member Functions

- std::string as_str ()

  string representation

Protected Member Functions

- virtual boost::any **_get_value** (const google::protobuf::FieldDescriptor *field, const google::protobuf::Message &msg)

- virtual boost::any **_get_repeated_value** (const google::protobuf::FieldDescriptor *field, const google::protobuf::Message &msg, int index)
- virtual void **_set_value** (const google::protobuf::FieldDescriptor *field, google::protobuf::Message *msg, boost::any value)
- virtual void **_add_value** (const google::protobuf::FieldDescriptor *field, google::protobuf::Message *msg, boost::any value)

17.23.1 Detailed Description

template<> class goby::acomms::FromProtoCppType< google::protobuf::FieldDescriptor::CPPTYPE_MESSAGE >

Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using the dynamic google::protobuf::Message as the underlying class.

Definition at line 496 of file protobuf_cpp_type_helpers.h.

The documentation for this class was generated from the following file:

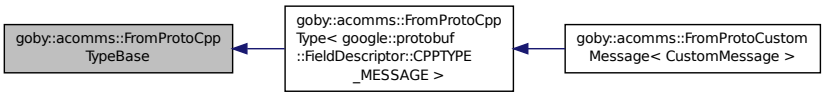
- goby/acomms/dccl/protobuf_cpp_type_helpers.h

17.24 goby::acomms::FromProtoCppTypeBase Class Reference

Provides various representations of a google::protobuf::FieldDescriptor::CppType enumeration, and ways to access the google::protobuf::Reflection object for a given type.

`#include <goby/acomms/dccl/protobuf_cpp_type_helpers.h>`

Inheritance diagram for goby::acomms::FromProtoCppTypeBase:



Public Member Functions

- virtual std::string as_str ()

  string representation
- boost::any get_value (const google::protobuf::FieldDescriptor *field, const google::protobuf::Message &msg)

  Get a given field's value from the provided message.
- boost::any get_value (const google::protobuf::Message &msg)

  Get the value of the entire base message (only works for CPPTYPE_MESSAGE)
- boost::any get_repeated_value (const google::protobuf::FieldDescriptor *field, const google::protobuf::Message &msg, int index)

  Get the value of a repeated field at a given index.
- void set_value (const google::protobuf::FieldDescriptor *field, google::protobuf::Message *msg, boost::any value)

  Set a given field's value in the provided message.
- void set_value (google::protobuf::Message *msg, boost::any value)

  Set the value of the entire base message (only works for CPPTYPE_MESSAGE)
- void add_value (const google::protobuf::FieldDescriptor *field, google::protobuf::Message *msg, boost::any value)

Add a new entry for a repeated field to the back.

- virtual void **_set_value** (const google::protobuf::FieldDescriptor *field, google::protobuf::Message *msg, boost::any value)
- virtual void **_add_value** (const google::protobuf::FieldDescriptor *field, google::protobuf::Message *msg, boost::any value)
- virtual boost::any **_get_repeated_value** (const google::protobuf::FieldDescriptor *field, const google::protobuf::Message &msg, int index)
- virtual boost::any **_get_value** (const google::protobuf::FieldDescriptor *field, const google::protobuf::Message &msg)

17.24.1 Detailed Description

Provides various representations of a google::protobuf::FieldDescriptor::CppType enumeration, and ways to access the google::protobuf::Reflection object for a given type.

Definition at line 158 of file protobuf_cpp_type_helpers.h.

17.24.2 Member Function Documentation

17.24.2.1 void goby::acomms::FromProtoCppTypeBase::add_value (const google::protobuf::FieldDescriptor * *field*, google::protobuf::Message * *msg*, boost::any *value*) [inline]

Add a new entry for a repeated field to the back.

Parameters

msg	Message to set value in.

Returns

boost::any containing the value. The type is usually the type returned by google::protobuf::Reflection::Add<i>Type where *Type* is the corresponding google::protobuf::FieldDescriptor::CppType. (See http://code.google.com/apis/protocolbuffers/docs/reference/cpp/google.protobuf.message.html#Reflection).

Definition at line 229 of file protobuf_cpp_type_helpers.h.

17.24.2.2 boost::any goby::acomms::FromProtoCppTypeBase::get_repeated_value (const google::protobuf::FieldDescriptor * *field*, const google::protobuf::Message & *msg*, int *index*) [inline]

Get the value of a repeated field at a given index.

Parameters

field	Field to get value for.
msg	Message to get value from.
index	Index of the repeated array to get value for.

Returns

boost::any containing the value. The type is usually the type returned by google::protobuf::Reflection::Get<i>Type where *Type* is the corresponding google::protobuf::FieldDescriptor::CppType. (See http://code.google.com/apis/protocolbuffers/docs/reference/cpp/google.protobuf.message.html#Reflection).

Definition at line 193 of file protobuf_cpp_type_helpers.h.

17.24.2.3 boost::any goby::acomms::FromProtoCppTypeBase::get_value (const google::protobuf::FieldDescriptor * *field*, const google::protobuf::Message & *msg*) [inline]

Get a given field's value from the provided message.

Parameters

field	Field to get value for.
msg	Message to get value from.

Returns

boost::any containing the value. The type is usually the type returned by google::protobuf::Reflection::Get<i>Type where *Type* is the corresponding google::protobuf::FieldDescriptor::CppType. (See http://code.google.com/apis/protocolbuffers/docs/reference/cpp/google.protobuf.message.html#Reflection).

Definition at line 171 of file protobuf_cpp_type_helpers.h.

17.24.2.4 void goby::acomms::FromProtoCppTypeBase::set_value (const google::protobuf::FieldDescriptor * *field*, google::protobuf::Message * *msg*, boost::any *value*) [inline]

Set a given field's value in the provided message.

Parameters

field	Field to set value for.
msg	Message to set value in.
value	boost::any containing the value to set. The type is usually the type required by google::protobuf::Reflection::Set<i>Type where *Type* is the corresponding google::protobuf::FieldDescriptor::CppType. (See http://code.google.com/apis/protocolbuffers/docs/reference/cpp/google.protobuf.message.html#Reflection).

Definition at line 205 of file protobuf_cpp_type_helpers.h.

The documentation for this class was generated from the following file:

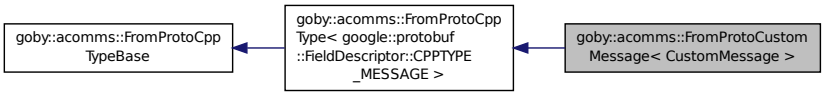
- goby/acomms/dccl/protobuf_cpp_type_helpers.h

17.25 goby::acomms::FromProtoCustomMessage< CustomMessage > Class Template Reference

Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using a specific statically generated Protobuf class.

`#include <goby/acomms/dccl/protobuf_cpp_type_helpers.h>`

Inheritance diagram for goby::acomms::FromProtoCustomMessage< CustomMessage >:



Public Types

- typedef CustomMessage **type**

Additional Inherited Members

17.25.1 Detailed Description

template<typename CustomMessage>class goby::acomms::FromProtoCustomMessage< CustomMessage >

Implements FromProtoCppTypeBase for CPPTYPE_MESSAGE using a specific statically generated Protobuf class.

Definition at line 566 of file protobuf_cpp_type_helpers.h.

The documentation for this class was generated from the following file:

- goby/acomms/dccl/protobuf_cpp_type_helpers.h

17.26 goby::acomms::FromProtoTypeBase Class Reference

Provides various representations of a google::protobuf::FieldDescriptor::Type enumeration. Implementations are provided for all enumerations.

`#include <goby/acomms/dccl/protobuf_cpp_type_helpers.h>`

Inherited by goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_BOOL >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_BYTES >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_DOUBLE >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_ENUM >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_FIXED32 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_FIXED64 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_FLOAT >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_GROUP >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_INT32 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_INT64 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_MESSAGE >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_SFIXED32 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_SFIXED64 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_SINT32 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_SINT64 >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_STRING >, goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_UINT32 >, and goby::acomms::FromProtoType< google::protobuf::FieldDescriptor::TYPE_UINT64 >.

Public Member Functions

- virtual std::string as_str ()

  string representation of the google::protobuf::FieldDescriptor::Type.

17.26.1 Detailed Description

Provides various representations of a google::protobuf::FieldDescriptor::Type enumeration. Implementations are provided for all enumerations.

Definition at line 38 of file protobuf_cpp_type_helpers.h.

The documentation for this class was generated from the following file:

- goby/acomms/dccl/protobuf_cpp_type_helpers.h

17.27 goby::acomms::MACManager Class Reference

provides an API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer.

`#include <goby/acomms/amac.h>`

Inherits std::list< T >.

Public Member Functions

Constructors/Destructor

- MACManager ()
  Default constructor.
- ~**MACManager** ()

Control

- void startup (const protobuf::MACConfig &cfg)
  Starts the MAC with given configuration.
- void restart ()
  Restarts the MAC with original configuration.
- void shutdown ()
  Shutdown the MAC.
- void do_work ()
  Allows the MAC timer to do its work. Does not block. If you prefer more control you can directly control the underlying boost::asio::io_service (get_io_service()) instead of using this function. This function is equivalent to get_io_service().poll();.
- void update ()
  You must call this after any change to the underlying list that would invalidate iterators or change the size (insert, push_back, erase, etc.).
- bool **running** ()

Modem Signals

- boost::signals2::signal< void(const protobuf::ModemTransmission &m)> signal_initiate_transmission
  Signals when it is time for this platform to begin transmission of an acoustic message at the start of its TDMA slot. Typically connected to ModemDriverBase::handle_initiate_transmission() using bind().
- unsigned **cycle_count** ()
- double **cycle_duration** ()
- boost::asio::io_service & **get_io_service** ()
- const std::string & **glog_mac_group** () const

Additional Inherited Members

17.27.1 Detailed Description

provides an API to the goby-acomms MAC library. MACManager is essentially a std::list<protobuf::ModemTransmission> plus a timer.

See Also

acomms_amac.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::acomms::protobuf).

Examples:

acomms/amac/amac_simple/amac_simple.cpp, and acomms/chat/chat.cpp.

Definition at line 56 of file mac_manager.h.

17.27.2 Member Function Documentation

17.27.2.1 void goby::acomms::MACManager::startup (const protobuf::MACConfig & *cfg*)

Starts the MAC with given configuration.

Parameters

cfg	Initial configuration values (protobuf::MACConfig defined in acomms_amac.proto)

Examples:

acomms/amac/amac_simple/amac_simple.cpp, and acomms/chat/chat.cpp.

Definition at line 74 of file mac_manager.cpp.

17.27.3 Member Data Documentation

17.27.3.1 boost::signals2::signal<void (const protobuf::ModemTransmission& m)> goby::acomms::MACManager::signal_initiate_transmission

Signals when it is time for this platform to begin transmission of an acoustic message at the start of its TDMA slot. Typically connected to ModemDriverBase::handle_initiate_transmission() using bind().

Parameters

m	a message containing details of the transmission to be initated. (protobuf::ModemMsgBase defined in acomms_modem_message.proto)

Examples:

acomms/amac/amac_simple/amac_simple.cpp, and acomms/chat/chat.cpp.

Definition at line 98 of file mac_manager.h.

The documentation for this class was generated from the following files:

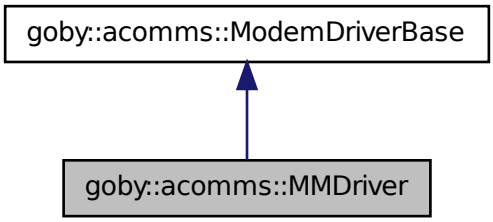
- goby/acomms/amac/mac_manager.h
- src/acomms/amac/mac_manager.cpp

17.28 goby::acomms::MMDriver Class Reference

provides an API to the WHOI Micro-Modem driver

```
#include <goby/acomms/modem_driver.h>
```

Inheritance diagram for goby::acomms::MMDriver:



Public Member Functions

- MMDriver ()
  Default constructor.
- ~MMDriver ()
  Destructor.
- void startup (const protobuf::DriverConfig &cfg)
  Starts the driver.
- void shutdown ()
  Stops the driver.
- void do_work ()
  See ModemDriverBase::do_work()
- void handle_initiate_transmission (const protobuf::ModemTransmission &m)
  See ModemDriverBase::handle_initiate_transmission()
- int clk_mode ()
  Current clock mode of the modem, necessary for synchronous navigation.
- bool **is_started** () const
- void **set_silent** (bool silent)

Static Public Member Functions

- static unsigned **packet_frame_count** (int rate)
- static unsigned **packet_size** (int rate)

Additional Inherited Members

17.28.1 Detailed Description

provides an API to the WHOI Micro-Modem driver

Examples:

acomms/chat/chat.cpp, and acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 42 of file mm_driver.h.

17.28.2 Member Function Documentation

17.28.2.1 void goby::acomms::MMDriver::startup (const protobuf::DriverConfig & *cfg*) [virtual]

Starts the driver.

Parameters

cfg	Configuration for the Micro-Modem driver. DriverConfig is defined in acomms_driver_base.proto, and various extensions specific to the WHOI Micro-Modem are defined in acomms_mm_driver.proto.

Implements goby::acomms::ModemDriverBase.

Examples:

acomms/chat/chat.cpp.

Definition at line 80 of file mm_driver.cpp.

The documentation for this class was generated from the following files:

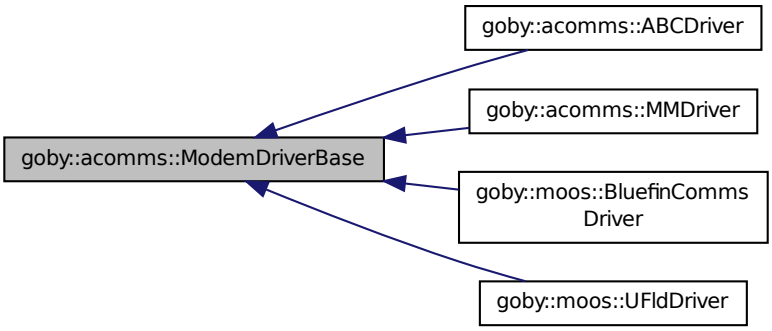
- goby/acomms/modemdriver/mm_driver.h
- src/acomms/modemdriver/mm_driver.cpp

17.29 goby::acomms::ModemDriverBase Class Reference

provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.

`#include <goby/acomms/modem_driver.h>`

Inheritance diagram for goby::acomms::ModemDriverBase:



Public Member Functions

- virtual ~ModemDriverBase ()

  Public Destructor.

Control

- virtual void startup (const protobuf::DriverConfig &cfg)=0
  Starts the modem driver. Must be called before poll().
- virtual void shutdown ()=0
  Shuts down the modem driver.
- virtual void do_work ()=0
  Allows the modem driver to do its work.

MAC Slots

- virtual void handle_initiate_transmission (const protobuf::ModemTransmission &m)=0
  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Public Attributes

MAC / Queue Signals

- boost::signals2::signal< void(const protobuf::ModemTransmission &message)> signal_receive
  Called when a binary data transmission is received from the modem.
- boost::signals2::signal< void(const protobuf::ModemTransmission &message)> signal_transmit_result
  Called when a transmission is completed.
- boost::signals2::signal< void(protobuf::ModemTransmission *msg)> signal_data_request
  Called when the modem or modem driver needs data to send. The returned data should be stored in ModemTransmission::frame.
- boost::signals2::signal< void(protobuf::ModemTransmission *msg_request)> signal_modify_transmission
  Called before the modem driver begins processing a transmission. This allows a third party to modify the parameters of the transmission (such as destination or rate) on the fly.
- boost::signals2::signal< void(const protobuf::ModemRaw &msg)> signal_raw_incoming
  Called after any message is received from the modem by the driver. Used by the MACManager for auto-discovery of vehicles. Also useful for higher level analysis and debugging of the transactions between the driver and the modem.
- boost::signals2::signal< void(const protobuf::ModemRaw &msg)> signal_raw_outgoing
  Called after any message is sent from the driver to the modem. Useful for higher level analysis and debugging of the transactions between the driver and the modem.

Protected Member Functions

Constructors/Destructor

- ModemDriverBase ()
  Constructor.

Write/read from the line-based interface to the modem

- void modem_write (const std::string &out)
  write a line to the serial port.
- bool modem_read (std::string *in)
  read a line from the serial port, including end-of-line character(s)
- void modem_start (const protobuf::DriverConfig &cfg)

start the physical connection to the modem (serial port, TCP, etc.). must be called before ModemDriverBase::modem_read() or ModemDriverBase::modem_write()

- void modem_close ()

  closes the serial port. Use modem_start to reopen the port.
- const std::string & **glog_out_group** () const
- const std::string & **glog_in_group** () const
- util::LineBasedInterface & modem ()

  use for direct access to the modem

Static Protected Attributes

- static int **count_** = 0

17.29.1 Detailed Description

provides an abstract base class for acoustic modem drivers. This is subclassed by the various drivers for different manufacturers' modems.

See Also

acomms_driver_base.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::acomms::protobuf).

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 46 of file driver_base.h.

17.29.2 Member Function Documentation

17.29.2.1 virtual void goby::acomms::ModemDriverBase::do_work ( ) [pure virtual]

Allows the modem driver to do its work.

Should be called regularly to perform the work of the driver as the driver *does not* run in its own thread. This allows us to guarantee that no signals are called except inside this method. Does not block.

Implemented in goby::acomms::MMDriver, goby::moos::BluefinCommsDriver, goby::moos::UFldDriver, and goby::acomms::ABCDriver.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

17.29.2.2 virtual void goby::acomms::ModemDriverBase::handle_initiate_transmission (const protobuf::ModemTransmission & *m*) [pure virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

m	ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)

Implemented in goby::acomms::MMDriver, goby::moos::BluefinCommsDriver, goby::moos::UFldDriver, and goby::acomms::ABCDriver.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

17.29.2.3 bool goby::acomms::ModemDriverBase::modem_read (std::string * *in*) [protected]

read a line from the serial port, including end-of-line character(s)

Parameters

in	pointer to string to store line

Returns

true if a line was available, false if no line available

Definition at line 72 of file driver_base.cpp.

17.29.2.4 void goby::acomms::ModemDriverBase::modem_start (const protobuf::DriverConfig & *cfg*) [protected]

start the physical connection to the modem (serial port, TCP, etc.). must be called before ModemDriverBase::modem_read() or ModemDriverBase::modem_write()

Parameters

cfg	Configuration including the parameters for the physical connection. (protobuf::DriverConfig is defined in acomms_driver_base.proto).

Exceptions

ModemDriverException	Problem opening the physical connection.

Definition at line 92 of file driver_base.cpp.

17.29.2.5 void goby::acomms::ModemDriverBase::modem_write (const std::string & *out*) [protected]

write a line to the serial port.

Parameters

out	reference to string to write. Must already include any end-of-line character(s).

Definition at line 60 of file driver_base.cpp.

17.29.2.6 virtual void goby::acomms::ModemDriverBase::startup (const protobuf::DriverConfig & *cfg*) [pure virtual]

Starts the modem driver. Must be called before poll().

Parameters

cfg	Startup configuration for the driver and modem. DriverConfig is defined in acomms_driver_base.proto. Derived classes can define extensions (see http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions) to DriverConfig to handle modem specific configuration.

Implemented in goby::acomms::MMDriver, goby::moos::BluefinCommsDriver, goby::moos::UFldDriver, and goby::acomms::ABCDriver.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

17.29.3 Member Data Documentation

17.29.3.1 boost::signals2::signal<void (protobuf::ModemTransmission* msg)> goby::acomms::ModemDriverBase::signal_data_request

Called when the modem or modem driver needs data to send. The returned data should be stored in ModemTransmission::frame.

You should connect one or more slots (a function or member function) to this signal to handle data requests. Use the goby::acomms::connect family of functions to do this. This signal will only be called during a call to poll. ModemTransmission is defined in acomms_modem_message.proto.

Definition at line 95 of file driver_base.h.

17.29.3.2 boost::signals2::signal<void (protobuf::ModemTransmission* msg_request)> goby::acomms::ModemDriverBase::signal_modify_transmission

Called before the modem driver begins processing a transmission. This allows a third party to modify the parameters of the transmission (such as destination or rate) on the fly.

You may connect one or more slots (a function or member function) to this signal to handle data requests. Use the goby::acomms::connect family of functions to do this. This signal will only be called during a call to poll. ModemTransmission is defined in acomms_modem_message.proto.

Definition at line 101 of file driver_base.h.

17.29.3.3 boost::signals2::signal<void (const protobuf::ModemRaw& msg)> goby::acomms::ModemDriverBase::signal_raw_incoming

Called after any message is received from the modem by the driver. Used by the MACManager for auto-discovery of vehicles. Also useful for higher level analysis and debugging of the transactions between the driver and the modem.

If desired, you should connect one or more slots (a function or member function) to this signal to listen on incoming transactions. Use the goby::acomms::connect family of functions to do this. This signal will only be called during a call to poll. ModemRaw is defined in acomms_modem_message.proto.

Definition at line 108 of file driver_base.h.

17.29.3.4 boost::signals2::signal<void (const protobuf::ModemRaw& msg)> goby::acomms::ModemDriverBase::signal_raw_outgoing

Called after any message is sent from the driver to the modem. Useful for higher level analysis and debugging of the transactions between the driver and the modem.

If desired, you should connect one or more slots (a function or member function) to this signal to listen on outgoing transactions. Use the goby::acomms::connect family of functions to do this. This signal will only be called during a call to poll. ModemRaw is defined in acomms_modem_message.proto.

Definition at line 114 of file driver_base.h.

17.29.3.5 boost::signals2::signal<void (const protobuf::ModemTransmission& message)> goby::acomms::ModemDriverBase::signal_receive

Called when a binary data transmission is received from the modem.

You should connect one or more slots (a function or member function) to this signal to receive incoming messages. Use the goby::acomms::connect family of functions to do this. This signal will only be called during a call to poll.

ModemDataTransmission is defined in acomms_modem_message.proto.

Examples:

acomms/modemdriver/driver_simple/driver_simple.cpp.

Definition at line 82 of file driver_base.h.

17.29.3.6 boost::signals2::signal<void (const protobuf::ModemTransmission& message)> goby::acomms::ModemDriverBase::signal_transmit_result

Called when a transmission is completed.

You should connect one or more slots (a function or member function) to this signal to receive incoming messages. Use the goby::acomms::connect family of functions to do this. This signal will only be called during a call to poll. ModemDataTransmission is defined in acomms_modem_message.proto.

Definition at line 89 of file driver_base.h.

The documentation for this class was generated from the following files:

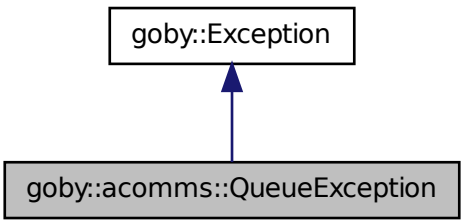
- goby/acomms/modemdriver/driver_base.h
- src/acomms/modemdriver/driver_base.cpp

17.30 goby::acomms::QueueException Class Reference

Exception class for libdccl.

`#include <goby/acomms/queue/queue_exception.h>`

Inheritance diagram for goby::acomms::QueueException:



Public Member Functions

- **QueueException** (const std::string &s)

17.30.1 Detailed Description

Exception class for libdccl.

Definition at line 36 of file queue_exception.h.

The documentation for this class was generated from the following file:

- goby/acomms/queue/queue_exception.h

17.31 goby::acomms::QueueManager Class Reference

provides an API to the goby-acomms Queuing Library.

`#include <goby/acomms/queue.h>`

Public Member Functions

- QueueManager ()
  constructor
- ~QueueManager ()
  destructor

Initialization Methods

These methods are intended to be called before doing any work with the class.

- void set_cfg (const protobuf::QueueManagerConfig &cfg)
  Set (and overwrite completely if present) the current configuration. (protobuf::QueueManagerConfig defined in acomms_queue.proto)
- void merge_cfg (const protobuf::QueueManagerConfig &cfg)
  Set (and merge "repeat" fields) the current configuration. (protobuf::QueueManagerConfig defined in acomms_queue.proto)
- template<typename ProtobufMessage >
  void add_queue (const protobuf::QueuedMessageEntry &queue_cfg)
  Add a DCCL queue for use with QueueManager. Note that the queue must be added before receiving messages with QueueManager.
- void add_queue (const google::protobuf::Descriptor *desc, const protobuf::QueuedMessageEntry &queue_cfg)
  Alternative method for adding Queues when using Dynamic Protobuf Messages.

Application level Push/Receive Methods

These methods are the primary higher level interface to the QueueManager. From here you can push messages and set the callbacks to use on received messages.

- void push_message (const google::protobuf::Message &new_message)
  Push a message (and add the queue if it does not exist)
- void **push_message** (const google::protobuf::Message &new_message, const protobuf::QueuedMessageMeta *meta)
- void flush_queue (const protobuf::QueueFlush &flush)
  Flush (delete all messages in) a queue.

Modem Slots

These methods are the interface to the QueueManager from the modem driver.

- void handle_modem_data_request (protobuf::ModemTransmission *msg)
  Finds data to send to the modem.
- void handle_modem_receive (const protobuf::ModemTransmission &message)
  Receive incoming data from the modem.

Control

Call these methods when you want the QueueManager to perform time sensitive tasks (such as expiring old messages)

- void do_work ()
  Calculates which messages have expired and emits the goby::acomms::QueueManager::signal_expire as necessary.

Informational Methods

- void info_all (std::ostream *os) const
  Writes a human readable summary (including DCCLCodec info) of all loaded queues.
- template<typename ProtobufMessage >
  void info (std::ostream *os) const
  Writes a human readable summary (including DCCLCodec info) of the queue for the provided DCCL type to the stream provided.
- void info (const google::protobuf::Descriptor *desc, std::ostream *os) const
  An alterative form for getting information for Queues for message types not known at compile-time ("dynamic").
- const std::string & **glog_push_group** ()
- const std::string & **glog_pop_group** ()
- const std::string & **glog_priority_group** ()
- const std::string & **glog_out_group** ()
- const std::string & **glog_in_group** ()
- std::string **msg_string** (const google::protobuf::Descriptor *desc)
- int modem_id ()
  The current modem ID (MAC address) of this node.
- protobuf::QueuedMessageMeta **meta_from_msg** (const google::protobuf::Message &msg)

Public Attributes

- boost::signals2::signal< void(protobuf::QueuedMessageMeta *meta, const google::protobuf::Message &data_msg, int modem_id)> signal_out_route
  Used by a router to change next-hop destination (in meta)
- boost::signals2::signal< void(const protobuf::QueuedMessageMeta &meta, const google::protobuf::Message &data_msg, int modem_id)> signal_in_route
  Used by a router to intercept messages and requeue them if desired.

Application Signals

- boost::signals2::signal< void(const protobuf::ModemTransmission &ack_msg, const google::protobuf::Message &orig_msg)> signal_ack
  Signals when acknowledgment of proper message receipt has been received. This is only sent for queues with queue.ack == true with an explicit destination (ModemMessageBase::dest() != 0)
- boost::signals2::signal< void(const google::protobuf::Message &msg) > signal_receive
  Signals when a DCCL message is received.
- boost::signals2::signal< void(const google::protobuf::Message &orig_msg)> signal_expire
  Signals when a message is expires (exceeds its time-to-live or ttl) before being sent (if queue.ack == false) or before being acknowledged (if queue.ack == true).
- boost::signals2::signal< void(const protobuf::ModemTransmission &request_msg, google::protobuf::Message *data_msg)> signal_data_on_demand
  Forwards the data request to the application layer. This advanced feature is used when queue.encode_on_demand == true and allows for the application to provide data immediately before it is actually sent (for highly time sensitive data)

- boost::signals2::signal< void(protobuf::QueueSize size)> signal_queue_size_change
  Signals when any queue changes size (message is popped or pushed)

Friends

- class **Queue**

17.31.1 Detailed Description

provides an API to the goby-acomms Queuing Library.

See Also

acomms_queue.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::acomms::protobuf).

Examples:

acomms/chat/chat.cpp, and acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 52 of file queue_manager.h.

17.31.2 Member Function Documentation

17.31.2.1 template<typename ProtobufMessage > void goby::acomms::QueueManager::add_queue (const protobuf::QueuedMessageEntry & *queue_cfg*) [inline]

Add a DCCL queue for use with QueueManager. Note that the queue must be added before receiving messages with QueueManager.

Template Parameters

ProtobufMessage	Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::Message)

Definition at line 76 of file queue_manager.h.

17.31.2.2 void goby::acomms::QueueManager::flush_queue (const protobuf::QueueFlush & *flush*)

Flush (delete all messages in) a queue.

Parameters

flush	QueueFlush object containing details about queue to flush

Definition at line 159 of file queue_manager.cpp.

17.31.2.3 void goby::acomms::QueueManager::handle_modem_data_request (protobuf::ModemTransmission * *msg*)

Finds data to send to the modem.

Data from the highest priority queue(s) will be combined to form a message equal or less than the size requested in ModemMessage message_in. If using one of the classes inheriting ModemDriverBase, this method should be connected to ModemDriverBase::signal_data_request.

Parameters

msg	The ModemTransmission containing information about the data request and is the place where the request data will be stored (in the repeated field ModemTransmission::frame).

Returns

true if successful in finding data to send, false if no data is available

Examples:

acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 213 of file queue_manager.cpp.

17.31.2.4 void goby::acomms::QueueManager::handle_modem_receive (const protobuf::ModemTransmission & *message*)

Receive incoming data from the modem.

If using one of the classes inheriting ModemDriverBase, this method should be bound and passed to ModemDriverBase::signal_receive.

Parameters

message	The received ModemMessage.

Examples:

acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 529 of file queue_manager.cpp.

17.31.2.5 template<typename ProtobufMessage > void goby::acomms::QueueManager::info (std::ostream * *os*) const [inline]

Writes a human readable summary (including DCCLCodec info) of the queue for the provided DCCL type to the stream provided.

Template Parameters

ProtobufMessage	Any Google Protobuf Message generated by protoc (i.e. subclass of google::protobuf::Message)

Parameters

os	Pointer to a stream to write this information

Definition at line 149 of file queue_manager.h.

17.31.2.6 void goby::acomms::QueueManager::info_all (std::ostream * *os*) const

Writes a human readable summary (including DCCLCodec info) of all loaded queues.

Parameters

os	Pointer to a stream to write this information

Definition at line 176 of file queue_manager.cpp.

17.31.2.7 void goby::acomms::QueueManager::push_message (const google::protobuf::Message & *new_message*)

Push a message (and add the queue if it does not exist)

Parameters

new_message	DCCL message to push.

Examples:

acomms/chat/chat.cpp, and acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 132 of file queue_manager.cpp.

17.31.3 Member Data Documentation

17.31.3.1 boost::signals2::signal<void (const protobuf::ModemTransmission& ack_msg, const google::protobuf::Message& orig_msg)> goby::acomms::QueueManager::signal_ack

Signals when acknowledgment of proper message receipt has been received. This is only sent for queues with queue.ack == true with an explicit destination (ModemMessageBase::dest() != 0)

Parameters

ack_msg	a message containing details of the acknowledgment and the acknowledged transmission. (protobuf::ModemMsgAck is defined in acomms_modem_message.proto)

Examples:

acomms/chat/chat.cpp.

Definition at line 191 of file queue_manager.h.

17.31.3.2 boost::signals2::signal<void (const protobuf::ModemTransmission& request_msg, google::protobuf::Message* data_msg)> goby::acomms::QueueManager::signal_data_on_demand

Forwards the data request to the application layer. This advanced feature is used when queue.encode_on_demand == true and allows for the application to provide data immediately before it is actually sent (for highly time sensitive data)

Parameters

request_msg	the details of the requested data. (protobuf::ModemDataRequest is defined in acomms_modem_message.proto)
data_msg	pointer to store the supplied data. The message is of the type for this queue.

Definition at line 208 of file queue_manager.h.

17.31.3.3 boost::signals2::signal<void (const google::protobuf::Message& orig_msg)> goby::acomms::QueueManager::signal_expire

Signals when a message is expires (exceeds its time-to-live or ttl) before being sent (if queue.ack == false) or before being acknowledged (if queue.ack == true).

Parameters

expire_msg	the expired transmission. (protobuf::ModemDataExpire is defined in acomms_modem_message.proto)

Definition at line 201 of file queue_manager.h.

17.31.3.4 boost::signals2::signal<void (protobuf::QueueSize size)> goby::acomms::QueueManager::signal_queue_size_change

Signals when any queue changes size (message is popped or pushed)

Parameters

size	message containing the queue that changed size and its new size (protobuf::QueueSize is defined in acomms_queue.proto).

Definition at line 213 of file queue_manager.h.

17.31.3.5 boost::signals2::signal<void (const google::protobuf::Message& msg) > goby::acomms::QueueManager::signal_receive

Signals when a DCCL message is received.

Parameters

msg	the received transmission.

Examples:

acomms/chat/chat.cpp, and acomms/queue/queue_simple/queue_simple.cpp.

Definition at line 196 of file queue_manager.h.

The documentation for this class was generated from the following files:

- goby/acomms/queue/queue_manager.h
- src/acomms/queue/queue_manager.cpp

17.32 goby::common::Colors Struct Reference

Represents the eight available terminal colors (and bold variants)

`#include <goby/common/logger/term_color.h>`

Public Types

- enum Color {
  nocolor, **red**, **lt_red**, **green**,
  lt_green, **yellow**, **lt_yellow**, **blue**,
  lt_blue, **magenta**, **lt_magenta**, **cyan**,
  lt_cyan, **white**, **lt_white** }

  The eight terminal colors (and bold or "light" variants)

17.32.1 Detailed Description

Represents the eight available terminal colors (and bold variants)

Definition at line 125 of file term_color.h.

The documentation for this struct was generated from the following file:

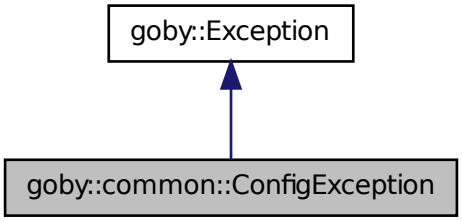
- goby/common/logger/term_color.h

17.33 goby::common::ConfigException Class Reference

indicates a problem with the runtime command line or .cfg file configuration (or –help was given)

`#include <goby/common/configuration_reader.h>`

Inheritance diagram for goby::common::ConfigException:



Public Member Functions

- **ConfigException** (const std::string &s)
- void **set_error** (bool b)
- bool **error** ()

17.33.1 Detailed Description

indicates a problem with the runtime command line or .cfg file configuration (or –help was given)

Definition at line 55 of file configuration_reader.h.

The documentation for this class was generated from the following file:

- goby/common/configuration_reader.h

17.34 goby::common::FlexNCurses Class Reference

Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.

`#include <goby/common/logger/flex_ncurses.h>`

Public Member Functions

Constructors / Destructor

- **FlexNCurses** ()
- ~**FlexNCurses** ()

Initialization

- void **startup** ()
- void **add_win** (const Group *title)

Use

- void insert (boost::posix_time::ptime t, const std::string &s, Group *g)

  Add a string to the gui.

Utility

- size_t **panel_from_group** (Group *g)
- void **recalculate_win** ()
- void **cleanup** ()
- void **alive** (bool alive)

User Input Thread

- void run_input ()

  run in its own thread to take input from the user

17.34.1 Detailed Description

Enables the Verbosity == gui mode of the Goby logger and displays an NCurses gui for the logger content.

Definition at line 44 of file flex_ncurses.h.

The documentation for this class was generated from the following files:

- goby/common/logger/flex_ncurses.h
- src/common/logger/flex_ncurses.cpp

17.35 goby::common::FlexOstream Class Reference

Forms the basis of the Goby logger: std::ostream derived class for holding the FlexOStreamBuf.

`#include <goby/common/logger/flex_ostream.h>`

Inherits std::ostream.

Public Member Functions

- **FlexOstream** (const FlexOstream &)
- FlexOstream & **operator=** (const FlexOstream &)
- void refresh ()

  Get a reference to the Goby logger mutex for scoped locking.
- void **set_group** (const std::string &s)

Initialization

- void add_group (const std::string &name, Colors::Color color=Colors::nocolor, const std::string &description="")
- void set_name (const std::string &s)

  Set the name of the application that the logger is serving.
- void **enable_gui** ()
- bool **is** (goby::common::logger::Verbosity verbosity, logger_lock::LockAction lock_action=logger_lock::none)
- void add_stream (logger::Verbosity verbosity=logger::VERBOSE, std::ostream *os=0)

  Attach a stream object (e.g. std::cout, std::ofstream, ...) to the logger with desired verbosity.
- void **add_stream** (goby::common::protobuf::GLogConfig::Verbosity verbosity=goby::common::protobuf::GLogConfig::VERBOSE, std::ostream *os=0)

Overloaded insert stream operator<<

- std::ostream & **operator**<< (FlexOstream &(*pf)(FlexOstream &))
- std::ostream & **operator**<< (std::ostream &(*pf)(std::ostream &))
- std::ostream & **operator**<< (bool &val)
- std::ostream & **operator**<< (const short &val)
- std::ostream & **operator**<< (const unsigned short &val)
- std::ostream & **operator**<< (const int &val)
- std::ostream & **operator**<< (const unsigned int &val)
- std::ostream & **operator**<< (const long &val)
- std::ostream & **operator**<< (const unsigned long &val)
- std::ostream & **operator**<< (const float &val)
- std::ostream & **operator**<< (const double &val)
- std::ostream & **operator**<< (const long double &val)
- std::ostream & **operator**<< (std::streambuf *sb)
- std::ostream & **operator**<< (std::ios &(*pf)(std::ios &))
- std::ostream & **operator**<< (std::ios_base &(*pf)(std::ios_base &))

Friends

- template<typename T >
  void **boost::checked_delete** (T *)
- std::ostream & **operator**<< (FlexOstream &out, char c)
- std::ostream & **operator**<< (FlexOstream &out, signed char c)
- std::ostream & **operator**<< (FlexOstream &out, unsigned char c)
- std::ostream & **operator**<< (FlexOstream &out, const char *s)
- std::ostream & **operator**<< (FlexOstream &out, const signed char *s)
- std::ostream & **operator**<< (FlexOstream &out, const unsigned char *s)

17.35.1 Detailed Description

Forms the basis of the Goby logger: std::ostream derived class for holding the FlexOStreamBuf.

Definition at line 54 of file flex_ostream.h.

17.35.2 Member Function Documentation

17.35.2.1 void goby::common::FlexOstream::add_group (const std::string & *name*, Colors::Color *color* = Colors::nocolor, const std::string & *description* = "")

Add another group to the logger. A group provides related manipulator for categorizing log messages. For thread safe use use boost::scoped_lock on Logger::mutex

Definition at line 44 of file flex_ostream.cpp.

The documentation for this class was generated from the following files:

- goby/common/logger/flex_ostream.h
- src/common/logger/flex_ostream.cpp

17.36 goby::common::FlexOStreamBuf Class Reference

Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by goby::common::FlexOstream for the Goby Logger (glogger)

`#include <goby/common/logger/flex_ostreambuf.h>`

Inherits streambuf.

Public Member Functions

- int sync ()

  virtual inherited from std::streambuf. Called when std::endl or std::flush is inserted into the stream
- int overflow (int c=EOF)

  virtual inherited from std::streambuf. Called when something is inserted into the stream
- void name (const std::string &s)

  name of the application being served
- void add_stream (logger::Verbosity verbosity, std::ostream *os)

  add a stream to the logger
- bool is_quiet ()

  do all attached streams have Verbosity == quiet?
- bool is_gui ()

  is there an attached stream with Verbosity == gui (ncurses GUI)
- void **enable_gui** ()
- logger::Verbosity **highest_verbosity** ()
- void group_name (const std::string &s)

  current group name (last insertion of group("") into the stream)
- void set_die_flag (bool b)

  exit on error at the next call to sync()
- void **set_verbosity_depth** (logger::Verbosity depth)
- void add_group (const std::string &name, Group g)

  add a new group
- std::string color2esc_code (Colors::Color color)

  TODO(tes): unnecessary?
- void refresh ()

  refresh the display (does nothing if !is_gui())

17.36.1 Detailed Description

Class derived from std::stringbuf that allows us to insert things before the stream and control output. This is the string buffer used by goby::common::FlexOstream for the Goby Logger (glogger)

Definition at line 75 of file flex_ostreambuf.h.

The documentation for this class was generated from the following files:

- goby/common/logger/flex_ostreambuf.h
- src/common/logger/flex_ostreambuf.cpp

17.37 goby::common::TermColor Class Reference

Converts between string, escape code, and enumeration representations of the terminal colors.

`#include <goby/common/logger/term_color.h>`

Static Public Member Functions

- static Colors::Color from_str (const std::string &s)

  Color enumeration from string (e.g. "blue" -> blue)
- static std::string str_from_col (const Colors::Color &c)

  String from color enumeration (e,g, red -> "red")
- static Colors::Color from_esc_code (const std::string &s)

Color enumeration from escape code (e.g, "\33[31m" -> red)

- static std::string esc_code_from_col (const Colors::Color &c)

  Escape code from color enumeration (e.g. red -> "\33[31m")

- static std::string esc_code_from_str (const std::string &s)

  Escape code from string (e.g. "red" -> "\33[31m")

Friends

- template<typename T >
  void **boost::checked_delete** (T *)

17.37.1 Detailed Description

Converts between string, escape code, and enumeration representations of the terminal colors.

Definition at line 139 of file term_color.h.

The documentation for this class was generated from the following files:

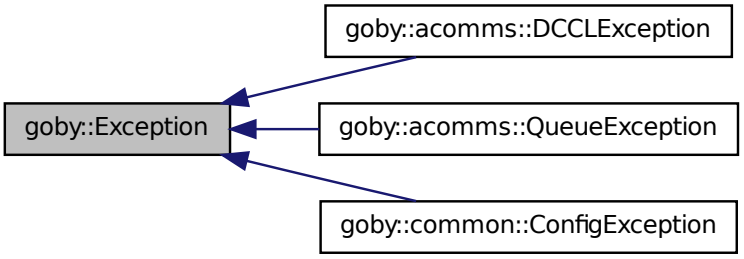
- goby/common/logger/term_color.h
- src/common/logger/term_color.cpp

17.38 goby::Exception Class Reference

simple exception class for goby applications

`#include <goby/common/exception.h>`

Inheritance diagram for goby::Exception:



Public Member Functions

- **Exception** (const std::string &s)

17.38.1 Detailed Description

simple exception class for goby applications

Definition at line 35 of file exception.h.

The documentation for this class was generated from the following file:

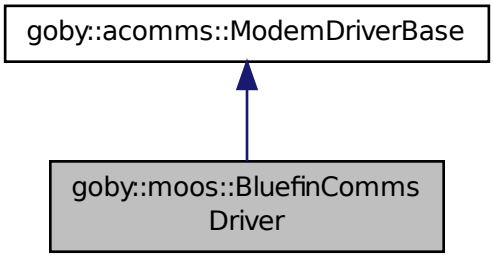
- goby/common/exception.h

17.39 goby::moos::BluefinCommsDriver Class Reference

provides a driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware)

`#include <goby/moos/moos_bluefin_driver.h>`

Inheritance diagram for goby::moos::BluefinCommsDriver:



Public Member Functions

- **BluefinCommsDriver** (goby::acomms::MACManager *mac)
- void startup (const goby::acomms::protobuf::DriverConfig &cfg)

  Starts the modem driver. Must be called before poll().
- void shutdown ()

  Shuts down the modem driver.
- void do_work ()

  Allows the modem driver to do its work.
- void handle_initiate_transmission (const goby::acomms::protobuf::ModemTransmission &m)

  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Additional Inherited Members

17.39.1 Detailed Description

provides a driver for the Bluefin Huxley communications infrastructure (initially uses SonarDyne as underlying hardware)

Definition at line 47 of file moos_bluefin_driver.h.

17.39.2 Member Function Documentation

17.39.2.1 void goby::moos::BluefinCommsDriver::do_work ( ) [virtual]

Allows the modem driver to do its work.

Should be called regularly to perform the work of the driver as the driver *does not* run in its own thread. This allows us to guarantee that no signals are called except inside this method. Does not block.

Implements goby::acomms::ModemDriverBase.

Definition at line 188 of file moos_bluefin_driver.cpp.

17.39.2.2 void goby::moos::BluefinCommsDriver::handle_initiate_transmission (const goby::acomms::protobuf::ModemTransmission & *m*) [virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

m	ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)

Implements goby::acomms::ModemDriverBase.

Definition at line 100 of file moos_bluefin_driver.cpp.

17.39.2.3 void goby::moos::BluefinCommsDriver::startup (const goby::acomms::protobuf::DriverConfig & *cfg*) [virtual]

Starts the modem driver. Must be called before poll().

Parameters

cfg	Startup configuration for the driver and modem. DriverConfig is defined in acomms_driver_base.proto. Derived classes can define extensions (see http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions) to DriverConfig to handle modem specific configuration.

Implements goby::acomms::ModemDriverBase.

Definition at line 54 of file moos_bluefin_driver.cpp.

The documentation for this class was generated from the following files:

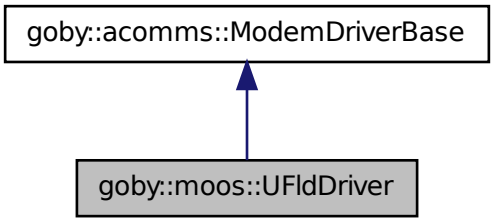
- goby/moos/moos_bluefin_driver.h
- src/moos/moos_bluefin_driver.cpp

17.40 goby::moos::UFldDriver Class Reference

provides an simulator driver to the uFldNodeComms MOOS module: http://oceanai.mit.edu/moos-ivp/pmwiki/pmwi
php?n=Modules.UFldNodeComms

```
#include <goby/moos/moos_ufield_sim_driver.h>
```

Inheritance diagram for goby::moos::UFldDriver:



Public Member Functions

- void startup (const goby::acomms::protobuf::DriverConfig &cfg)

  Starts the modem driver. Must be called before poll().
- void shutdown ()

  Shuts down the modem driver.
- void do_work ()

  Allows the modem driver to do its work.
- void handle_initiate_transmission (const goby::acomms::protobuf::ModemTransmission &m)

  Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Additional Inherited Members

17.40.1 Detailed Description

provides an simulator driver to the uFldNodeComms MOOS module: http://oceanai.mit.edu/moos-ivp/pmwiki/pmwi
php?n=Modules.UFldNodeComms

Definition at line 45 of file moos_ufield_sim_driver.h.

17.40.2 Member Function Documentation

17.40.2.1 void goby::moos::UFldDriver::do_work ( ) [virtual]

Allows the modem driver to do its work.

Should be called regularly to perform the work of the driver as the driver *does not* run in its own thread. This allows us to guarantee that no signals are called except inside this method. Does not block.

Implements goby::acomms::ModemDriverBase.

Definition at line 209 of file moos_ufield_sim_driver.cpp.

17.40.2.2 void goby::moos::UFldDriver::handle_initiate_transmission (const goby::acomms::protobuf::ModemTransmission & *m*) [virtual]

Virtual initiate_transmission method. Typically connected to MACManager::signal_initiate_transmission() using bind().

Parameters

m	ModemTransmission (defined in acomms_modem_message.proto) containing the details of the transmission to be started. This may contain data frames. If not, data will be requested when the driver calls the data request signal (ModemDriverBase::signal_data_request)

Implements goby::acomms::ModemDriverBase.

Definition at line 91 of file moos_ufield_sim_driver.cpp.

17.40.2.3 void goby::moos::UFldDriver::startup (const goby::acomms::protobuf::DriverConfig & *cfg*) [virtual]

Starts the modem driver. Must be called before poll().

Parameters

cfg	Startup configuration for the driver and modem. DriverConfig is defined in acomms_driver_base.proto. Derived classes can define extensions (see http://code.google.com/apis/protocolbuffers/docs/proto.html#extensions) to DriverConfig to handle modem specific configuration.

Implements goby::acomms::ModemDriverBase.

Definition at line 47 of file moos_ufield_sim_driver.cpp.

The documentation for this class was generated from the following files:

- goby/moos/moos_ufield_sim_driver.h
- src/moos/moos_ufield_sim_driver.cpp

17.41 goby::pb::Application Class Reference

Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture.

`#include <goby/pb/application.h>`

Inherits goby::common::ZeroMQApplicationBase.

Inherited by goby::acomms::Bridge, goby::acomms::FileTransfer, and goby::acomms::ModemDriver.

Protected Types

- typedef goby::common::Colors **Colors**

Protected Member Functions

- template<typename ProtoBufMessage >
  void subscribe (boost::function< void(const ProtoBufMessage &)> handler=boost::function< void(const ProtoBufMessage &)>(), const std::string &group="")

  Subscribe to a message (of any type derived from google::protobuf::Message)
- template<typename ProtoBufMessage , class C >
  void subscribe (void(C::*mem_func)(const ProtoBufMessage &), C *obj, const std::string &group="")

  Subscribe for a type using a class member function as the handler.
- common::ZeroMQService & zeromq_service ()

  Fetchs the newest received message of this type.

Constructors / Destructor

- Application (google::protobuf::Message *cfg=0)

- virtual ~**Application** ()

Publish / Subscribe

- void **publish** (const google::protobuf::Message &msg, const std::string &group="")

Additional Inherited Members

17.41.1 Detailed Description

Base class provided for users to generate applications that participate in the Goby publish/subscribe architecture.

Definition at line 47 of file application.h.

17.41.2 Constructor & Destructor Documentation

17.41.2.1 goby::pb::Application::Application (google::protobuf::Message * *cfg* = 0) [protected]

Parameters

cfg	pointer to object derived from google::protobuf::Message that defines the configuration for this Application. This constructor will use the Description of `cfg` to read the command line parameters and configuration file (if given) and use these values to populate `cfg`. `cfg` must be a static member of the subclass or global object since member objects will be constructed *after* the Application constructor is called.

Definition at line 40 of file application.cpp.

17.41.3 Member Function Documentation

17.41.3.1 template<typename ProtoBufMessage > void goby::pb::Application::subscribe (boost::function< void(const ProtoBufMessage &)> *handler* = `boost::function<void (const ProtoBufMessage&)>()`, const std::string & *group* = " ") [inline], [protected]

Subscribe to a message (of any type derived from google::protobuf::Message)

Parameters

handler	Function object to be called as soon as possible upon receipt of a message of this type. The signature of `handler` must match: void handler(const ProtoBufMessage& msg). if `handler` is omitted, no handler is called and only the newest message buffer is updated upon message receipt (for calls to newest<ProtoBufMessage>())

Definition at line 73 of file application.h.

17.41.3.2 template<typename ProtoBufMessage , class C > void goby::pb::Application::subscribe (void(C::*)(const ProtoBufMessage &) *mem_func*, C * *obj*, const std::string & *group* = " ") [inline], [protected]

Subscribe for a type using a class member function as the handler.

Parameters

mem_func	Member function (method) of class C with a signature of void C::mem_func(const ProtoBufMessage& msg)
obj	pointer to the object whose member function (mem_func) to call

Definition at line 89 of file application.h.

17.41.3.3 common::ZeroMQService& goby::pb::Application::zeromq_service ( ) [inline], [protected]

Fetchs the newest received message of this type.

You must subscribe() for this type before using this method

Definition at line 113 of file application.h.

The documentation for this class was generated from the following files:

- goby/pb/application.h
- src/pb/application.cpp

17.42 goby::transitional::DCCLMessageVal Class Reference

defines a DCCL value

`#include <goby/moos/transitional/message_val.h>`

Public Types

- enum { **MAX_DBL_PRECISION** = 15 }

Public Member Functions

Constructors/Destructor

- DCCLMessageVal ()
  empty
- DCCLMessageVal (const std::string &s)
  construct with string value
- DCCLMessageVal (const char *s)
  construct with char value*
- DCCLMessageVal (double d, int p=MAX_DBL_PRECISION)
  construct with double value, optionally givig the precision of the double (number of decimal places) which is used if a cast to std::string is required in the future.
- DCCLMessageVal (long l)
  construct with long value
- DCCLMessageVal (int i)
  construct with int value
- DCCLMessageVal (float f)
  construct with float value
- DCCLMessageVal (bool b)
  construct with bool value
- DCCLMessageVal (const std::vector< DCCLMessageVal > &vm)
  construct with vector

Setters

- void set (std::string sval)
  set the value with a string (overwrites previous value regardless of type)
- void set (double dval, int precision=MAX_DBL_PRECISION)
  set the value with a double (overwrites previous value regardless of type)
- void set (long lval)
  set the value with a long (overwrites previous value regardless of type)
- void set (bool bval)
  set the value with a bool (overwrites previous value regardless of type)

Getters

- bool get (std::string &s) const
  extract as std::string (all reasonable casts are done)
- bool get (bool &b) const
  extract as bool (all reasonable casts are done)
- bool get (long &t) const
  extract as long (all reasonable casts are done)
- bool get (double &d) const
  extract as double (all reasonable casts are done)
- operator double () const
- operator bool () const
- operator std::string () const
- operator long () const
- operator int () const
- operator unsigned () const
- operator float () const
- **operator std::vector**< **DCCLMessageVal** > () const
- DCCLCppType type () const
  what type is the original type of this DCCLMessageVal?
- bool empty () const
  was this just constructed with DCCLMessageVal() ?
- unsigned **precision** () const

Comparison

- bool **operator==** (const DCCLMessageVal &mv) const
- bool **operator==** (const std::string &s) const
- bool **operator==** (double d) const
- bool **operator==** (long l) const
- bool **operator==** (bool b) const

Friends

- std::ostream & **operator**<< (std::ostream &os, const DCCLMessageVal &mv)

17.42.1 Detailed Description

defines a DCCL value

Definition at line 38 of file message_val.h.

17.42.2 Member Function Documentation

17.42.2.1 bool goby::transitional::DCCLMessageVal::get (std::string & *s*) const

extract as std::string (all reasonable casts are done)

Parameters

s	std::string to store value in

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 129 of file message_val.cpp.

17.42.2.2 bool goby::transitional::DCCLMessageVal::get (bool & *b*) const

extract as bool (all reasonable casts are done)

Parameters

b	bool to store value in

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 160 of file message_val.cpp.

17.42.2.3 bool goby::transitional::DCCLMessageVal::get (long & *t*) const

extract as long (all reasonable casts are done)

Parameters

t	long to store value in

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 192 of file message_val.cpp.

17.42.2.4 bool goby::transitional::DCCLMessageVal::get (double & *d*) const

extract as double (all reasonable casts are done)

Parameters

d	double to store value in

Returns

successfully extracted (and if necessary successfully cast to this type)

Definition at line 231 of file message_val.cpp.

17.42.2.5 goby::transitional::DCCLMessageVal::operator bool ( ) const

allows statements of the form

```
bool b = DCCLMessageVal("1");
```

Definition at line 278 of file message_val.cpp.

17.42.2.6 goby::transitional::DCCLMessageVal::operator double ( ) const

allows statements of the form

```
double d = DCCLMessageVal("3.23");
```

Definition at line 270 of file message_val.cpp.

17.42.2.7 goby::transitional::DCCLMessageVal::operator float ( ) const

allows statements of the form

```
float f = DCCLMessageVal("3.5");
```

Definition at line 309 of file message_val.cpp.

17.42.2.8 goby::transitional::DCCLMessageVal::operator int ( ) const

allows statements of the form

```
int i = DCCLMessageVal(2);
```

Definition at line 299 of file message_val.cpp.

17.42.2.9 goby::transitional::DCCLMessageVal::operator long ( ) const

allows statements of the form

```
long l = DCCLMessageVal(5);
```

Definition at line 292 of file message_val.cpp.

17.42.2.10 goby::transitional::DCCLMessageVal::operator std::string ( ) const

allows statements of the form

```
std::string s = DCCLMessageVal(3);
```

Definition at line 285 of file message_val.cpp.

17.42.2.11 goby::transitional::DCCLMessageVal::operator unsigned ( ) const

allows statements of the form

```
unsigned u = DCCLMessageVal(2);
```

Definition at line 304 of file message_val.cpp.

17.42.2.12 void goby::transitional::DCCLMessageVal::set (double *dval*, int *precision* = MAX_DBL_PRECISION)

set the value with a double (overwrites previous value regardless of type)

Parameters

dval	values to set
precision	decimal places of precision to preserve if this is cast to a string

Definition at line 121 of file message_val.cpp.

The documentation for this class was generated from the following files:

- goby/moos/transitional/message_val.h
- src/moos/transitional/message_val.cpp

17.43 goby::transitional::DCCLTransitionalCodec Class Reference

provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

`#include <goby/acomms/dccl.h>`

Public Member Functions

- template<typename Key >
  const
  google::protobuf::Descriptor * **descriptor** (const Key &k)
- unsigned message_count ()
- template<typename Key >
  unsigned get_repeat (const Key &k)
- std::set< unsigned > all_message_ids ()
- std::set< std::string > all_message_names ()
- template<typename Key >
  std::map< std::string,
  std::string > message_var_names (const Key &k) const
- std::string id2name (unsigned id)
- unsigned name2id (const std::string &name)
- std::vector< DCCLMessage > & **messages** ()

Constructors/Destructor

- DCCLTransitionalCodec ()
  Instantiate optionally with a ostream logger (for human readable output)
- ~DCCLTransitionalCodec ()
  destructor

Initialization Methods.

These methods are intended to be called before doing any work with the class. However, they may be called at any time as desired.

- void **convert_to_v2_representation** (pAcommsHandlerConfig *cfg)

17.43.1 Detailed Description

provides an API to the Transitional Dynamic CCL Codec (looks like DCCLv1, but calls DCCLv2). Warning: this class is for legacy support only, new applications should use DCCLCodec directly.

See Also

transitional.proto and acomms_modem_message.proto for definition of Google Protocol Buffers messages (namespace goby::transitional::protobuf).

Definition at line 99 of file dccl_transitional.h.

17.43.2 Constructor & Destructor Documentation

17.43.2.1 goby::transitional::DCCLTransitionalCodec::DCCLTransitionalCodec ( )

Instantiate optionally with a ostream logger (for human readable output)

Parameters

log	std::ostream object or FlexOstream to capture all humanly readable runtime and debug information (optional).

Definition at line 46 of file dccl_transitional.cpp.

17.43.3 Member Function Documentation

17.43.3.1 std::set<unsigned> goby::transitional::DCCLTransitionalCodec::all_message_ids ( )

Returns

set of all message ids loaded

17.43.3.2 std::set<std::string> goby::transitional::DCCLTransitionalCodec::all_message_names ( )

Returns

set of all message names loaded

17.43.3.3 template<typename Key > unsigned goby::transitional::DCCLTransitionalCodec::get_repeat (const Key & *k*) [inline]

Returns

repeat value (number of copies of the message per encode)

Definition at line 134 of file dccl_transitional.h.

17.43.3.4 std::string goby::transitional::DCCLTransitionalCodec::id2name (unsigned *id*) [inline]

Parameters

id	message id

Returns

name of message

Definition at line 148 of file dccl_transitional.h.

17.43.3.5 unsigned goby::transitional::DCCLTransitionalCodec::message_count ( ) [inline]

how many message are loaded?

Returns

number of messages loaded

Definition at line 130 of file dccl_transitional.h.

17.43.3.6 template<typename Key > std::map<std::string, std::string> goby::transitional::DCCLTransitionalCodec::message_var_names (const Key & *k*) const [inline]

Returns

map of names to DCCL types needed to encode a given message

Definition at line 143 of file dccl_transitional.h.

17.43.3.7 unsigned goby::transitional::DCCLTransitionalCodec::name2id (const std::string & *name*) [inline]

Parameters

name	message name

Returns

id of message

Definition at line 151 of file dccl_transitional.h.

The documentation for this class was generated from the following files:

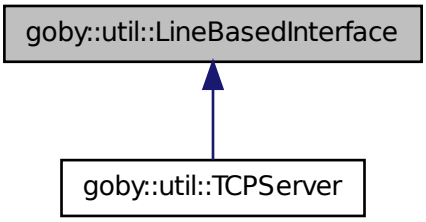
- goby/moos/transitional/dccl_transitional.h
- src/moos/transitional/dccl_transitional.cpp

17.44 goby::util::LineBasedInterface Class Reference

basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)

`#include <goby/util/linebasedcomms/interface.h>`

Inheritance diagram for goby::util::LineBasedInterface:



Public Types

- enum **AccessOrder** { **NEWEST_FIRST**, **OLDEST_FIRST** }

Public Member Functions

- **LineBasedInterface** (const std::string &delimiter)
- void **start** ()
- void **close** ()
- bool **active** ()
- bool readline (std::string ∗s, AccessOrder order=OLDEST_FIRST)

  returns string line (including delimiter)
- bool **readline** (protobuf::Datagram ∗msg, AccessOrder order=OLDEST_FIRST)
- void **write** (const std::string &s)
- void **write** (const protobuf::Datagram &msg)
- void **clear** ()
- void **set_delimiter** (const std::string &s)
- std::string **delimiter** () const
- boost::asio::io_service & **io_service** ()

Protected Member Functions

- virtual void **do_start** ()=0
- virtual void **do_write** (const protobuf::Datagram &line)=0
- virtual void **do_close** (const boost::system::error_code &error)=0
- void **set_active** (bool active)
- std::string & **delimiter** ()
- std::deque
  < goby::util::protobuf::Datagram > & **in** ()
- boost::mutex & **in_mutex** ()

Protected Attributes

- std::string **delimiter_**
- boost::asio::io_service **io_service_**
- std::deque< protobuf::Datagram > **in_**
- boost::mutex **in_mutex_**

Friends

- class **LineBasedConnection**

17.44.1 Detailed Description

basic interface class for all the derived serial (and networking mimics) line-based nodes (serial, tcp, udp, etc.)

Definition at line 48 of file interface.h.

17.44.2 Member Function Documentation

17.44.2.1 bool goby::util::LineBasedInterface::readline (std::string * *s*, AccessOrder *order* = OLDEST_FIRST) [inline]

returns string line (including delimiter)

Returns

true if data was read, false if no data to read

Definition at line 68 of file interface.h.

The documentation for this class was generated from the following files:

- goby/util/linebasedcomms/interface.h
- src/util/linebasedcomms/interface.cpp

17.45 goby::util::SerialClient Class Reference

provides a basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control

`#include <goby/util/linebasedcomms/serial_client.h>`

Inherits goby::util::LineBasedClient< ASIOAsyncReadStream >.

Public Member Functions

- SerialClient (const std::string &name="", unsigned baud=9600, const std::string &delimiter="\r\n")
  create a serial client
- void set_name (const std::string &name)
  set serial port name, e.g. "/dev/ttyS0"
- void set_baud (unsigned baud)
  baud rate, e.g. 4800
- std::string name () const
  serial port name, e.g. "/dev/ttyS0"
- unsigned baud () const
  baud rate, e.g. 4800
- boost::asio::serial_port & **socket** ()
- std::string local_endpoint ()
  our serial port, e.g. "/dev/ttyUSB1"
- std::string remote_endpoint ()
  who knows where the serial port goes?! (empty string)

Additional Inherited Members

17.45.1 Detailed Description

provides a basic client for line by line text based communications over a 8N1 tty (such as an RS-232 serial link) without flow control

Definition at line 37 of file serial_client.h.

17.45.2 Constructor & Destructor Documentation

17.45.2.1 goby::util::SerialClient::SerialClient (const std::string & *name* = "", unsigned *baud* = 9600, const std::string & *delimiter* = "\r\n")

create a serial client

Parameters

name	name of the serial connection (e.g. "/dev/ttyS0")
baud	baud rate of the serial connection (e.g. 9600)
delimiter	string used to split lines

Definition at line 32 of file serial_client.cpp.

The documentation for this class was generated from the following files:

- goby/util/linebasedcomms/serial_client.h
- src/util/linebasedcomms/serial_client.cpp

17.46 goby::util::TCPClient Class Reference

provides a basic TCP client for line by line text based communications to a remote TCP server

`#include <goby/util/linebasedcomms/tcp_client.h>`

Inherits goby::util::LineBasedClient< ASIOAsyncReadStream >.

Public Member Functions

- TCPClient (const std::string &server, unsigned port, const std::string &delimiter="\r\n", int retry_interval=10)

  create a TCPClient
- boost::asio::ip::tcp::socket & **socket** ()
- std::string local_endpoint ()

  string representation of the local endpoint (e.g. 192.168.1.105:54230
- std::string remote_endpoint ()

  string representation of the remote endpoint, (e.g. 192.168.1.106:50000

Friends

- class **TCPConnection**
- class **LineBasedConnection**< **boost::asio::ip::tcp::socket** >

Additional Inherited Members

17.46.1 Detailed Description

provides a basic TCP client for line by line text based communications to a remote TCP server

Definition at line 36 of file tcp_client.h.

17.46.2 Constructor & Destructor Documentation

17.46.2.1 goby::util::TCPClient::TCPClient (const std::string & *server*, unsigned *port*, const std::string & *delimiter* = "\r\n", int *retry_interval* = 10)

create a TCPClient

Parameters

server	domain name or IP address of the remote server
port	port of the remote server
delimiter	string used to split lines

Definition at line 29 of file tcp_client.cpp.

The documentation for this class was generated from the following files:

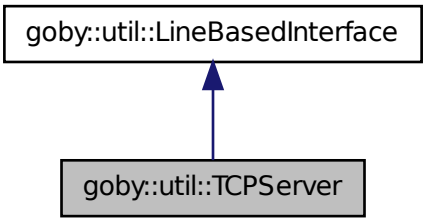
- goby/util/linebasedcomms/tcp_client.h
- src/util/linebasedcomms/tcp_client.cpp

17.47 goby::util::TCPServer Class Reference

provides a basic TCP server for line by line text based communications to a one or more remote TCP clients

```
#include <goby/util/linebasedcomms/tcp_server.h>
```

Inheritance diagram for goby::util::TCPServer:



Public Member Functions

- TCPServer (unsigned port, const std::string &delimiter="\r\n")
  create a TCP server
- std::string local_endpoint ()
  string representation of the local endpoint (e.g. 192.168.1.105:54230
- const std::set
  < boost::shared_ptr
  < TCPConnection > > & **connections** ()

Friends

- class **TCPConnection**
- class **LineBasedConnection**< **boost::asio::ip::tcp::socket** >

Additional Inherited Members

17.47.1 Detailed Description

provides a basic TCP server for line by line text based communications to a one or more remote TCP clients

Definition at line 52 of file tcp_server.h.

17.47.2 Constructor & Destructor Documentation

17.47.2.1 goby::util::TCPServer::TCPServer (unsigned *port*, const std::string & *delimiter* = "\r\n") [inline]

create a TCP server

Parameters

port	port of the server (use 50000+ to avoid problems with special system ports)
delimiter	string used to split lines

Definition at line 59 of file tcp_server.h.

The documentation for this class was generated from the following files:

- goby/util/linebasedcomms/tcp_server.h

- src/util/linebasedcomms/tcp_server.cpp

17.48 Group Class Reference

Defines a group of messages to be sent to the Goby logger. For Verbosity == verbose streams, all entries appear interleaved, but each group is offset with a different color. For Verbosity == gui streams, all groups have a separate subwindow.

`#include <goby/common/logger/logger_manipulators.h>`

Public Member Functions

- **Group** (const std::string &name="", const std::string &description="", goby::common::Colors::Color color=goby::common::Colors::nocolor)

Getters

- std::string name () const
  Name of this group (used in the group manipulator)
- std::string description () const
  Human readable description of this group.
- goby::common::Colors::Color color () const
  Color to use when displaying this group (for streams that support terminal escape codes only: std::cout, std::cerr, std::clog)
- bool enabled () const
  Is this group enabled?

Setters

- void **name** (const std::string &s)
- void **description** (const std::string &s)
- void **color** (goby::common::Colors::Color c)
- void **enabled** (bool b)

17.48.1 Detailed Description

Defines a group of messages to be sent to the Goby logger. For Verbosity == verbose streams, all entries appear interleaved, but each group is offset with a different color. For Verbosity == gui streams, all groups have a separate subwindow.

Definition at line 70 of file logger_manipulators.h.

The documentation for this class was generated from the following file:

- goby/common/logger/logger_manipulators.h

17.49 GroupSetter Class Reference

Helper class for enabling the group(std::string) manipulator.

`#include <goby/common/logger/logger_manipulators.h>`

Public Member Functions

- **GroupSetter** (const std::string &s)
- void **operator()** (std::ostream &os) const
- void **operator()** (goby::common::FlexOstream &os) const

17.49.1 Detailed Description

Helper class for enabling the group(std::string) manipulator.

Definition at line 117 of file logger_manipulators.h.

The documentation for this class was generated from the following files:

- goby/common/logger/logger_manipulators.h
- src/common/logger/logger_manipulators.cpp

18 File Documentation

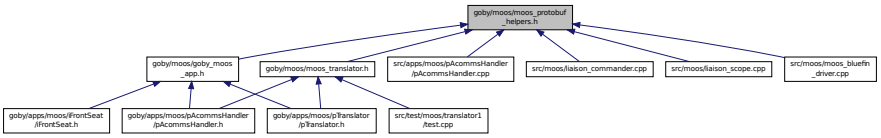
18.1 goby/moos/moos_protobuf_helpers.h File Reference

Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages.

```
#include <limits>
#include <boost/format.hpp>
#include <boost/regex.hpp>
#include <google/protobuf/io/printer.h>
#include <google/protobuf/io/tokenizer.h>
#include "goby/common/logger/flex_ostream.h"
#include "goby/util/as.h"
#include "goby/util/binary.h"
#include "goby/util/dynamic_protobuf_manager.h"
#include "goby/moos/moos_string.h"
#include "goby/util/primitive_types.h"
#include "goby/moos/transitional/message_algorithms.h"
#include "goby/moos/transitional/message_val.h"
#include "goby/moos/protobuf/translator.pb.h"
```

Include dependency graph for moos_protobuf_helpers.h:

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace goby

  The global namespace for the Goby project.

Functions

- std::map< int, std::string > **goby::moos::run_serialize_algorithms** (const google::protobuf::Message &in, const google::protobuf::RepeatedPtrField< protobuf::TranslatorEntry::PublishSerializer::Algorithm > &algorithms)
- std::string **goby::moos::strip_name_from_enum** (const std::string &enum_value, const std::string &field_name)
- std::string **goby::moos::add_name_to_enum** (const std::string &enum_value, const std::string &field_name)
- void serialize_for_moos (std::string *out, const google::protobuf::Message &msg)

  Converts the Google Protocol Buffers message `msg` into a suitable (human readable) string `out` for sending via MOOS.
- void parse_for_moos (const std::string &in, google::protobuf::Message *msg)

  Parses the string `in` to Google Protocol Buffers message `msg`. All errors are written to the goby::util::glogger().
- boost::shared_ptr < google::protobuf::Message > **dynamic_parse_for_moos** (const std::string &in)

Variables

- const std::string **goby::moos::MAGIC_PROTOBUF_HEADER** = "@PB"

18.1.1 Detailed Description

Helpers for MOOS applications for serializing and parsed Google Protocol buffers messages.

Definition in file moos_protobuf_helpers.h.

18.1.2 Function Documentation

18.1.2.1 void parse_for_moos (const std::string & *in*, google::protobuf::Message * *msg*) [inline]

Parses the string `in` to Google Protocol Buffers message `msg`. All errors are written to the goby::util::glogger().

Parameters

in	std::string to parse
msg	Google Protocol buffers message to store result

Definition at line 1286 of file moos_protobuf_helpers.h.

18.1.2.2 void serialize_for_moos (std::string * *out*, const google::protobuf::Message & *msg*) [inline]

Converts the Google Protocol Buffers message `msg` into a suitable (human readable) string `out` for sending via MOOS.

Parameters

out	pointer to std::string to store serialized result
msg	Google Protocol buffers message to serialize

Definition at line 1277 of file moos_protobuf_helpers.h.

19 Example Documentation

19.1 acomms/amac/amac_simple/amac_simple.cpp

```
#include "goby/acomms/amac.h"
#include "goby/acomms/connect.h"
#include "goby/common/logger.h"

#include <iostream>

using goby::acomms::operator<<;

void init_transmission(const goby::acomms::protobuf::ModemTransmission& msg);

int main(int argc, char* argv[])
{
    goby::glog.set_name(argv[0]);
    goby::glog.add_stream(goby::common::logger::DEBUG1, &
      std::clog);

    //
    // 1. Create a MACManager
    //
    goby::acomms::MACManager mac;

    //
    // 2. Configure with a few slots
    //
    goby::acomms::protobuf::MACConfig cfg;
    cfg.set_modem_id(1);
    cfg.set_type(goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED);
    goby::acomms::protobuf::ModemTransmission* slot = cfg.add_slot();
    slot->set_src(1);
    slot->set_rate(0);
    slot->set_type(goby::acomms::protobuf::ModemTransmission::DATA);
    slot->set_slot_seconds(5);


    //
    // 3. Set up the callback
    //

    // give a callback to use for actually initiating the transmission. this
       would be bound to goby::acomms::ModemDriverBase::initiate_transmission if using
       libmodemdriver.
    goby::acomms::connect(&mac.
      signal_initiate_transmission, &init_transmission);

    //
    // 4. Let it run for a bit alone in the world
    //
    mac.startup(cfg);
    for(unsigned i = 1; i < 150; ++i)
    {
        mac.do_work();
        usleep(100000);
    }

    //
    // 5. Add some slots (modem ids 2 & 3, and LBL ping from 1): remember
       MACManager is a std::list<goby::acomms::protobuf::ModemTransmission> at heart
    //

    // one way we can do this
    goby::acomms::protobuf::ModemTransmission new_slot;
    new_slot.set_src(2);
    new_slot.set_rate(0);
    new_slot.set_type(goby::acomms::protobuf::ModemTransmission::DATA);
    new_slot.set_slot_seconds(5);

    mac.push_back(new_slot);

    // another way
    mac.resize(mac.size()+1);
    mac.back().CopyFrom(mac.front());
    mac.back().set_src(3);

    mac.resize(mac.size()+1);
    mac.back().CopyFrom(mac.front());
    mac.back().set_type(
      goby::acomms::protobuf::ModemTransmission::DRIVER_SPECIFIC);
    mac.back().SetExtension(micromodem::protobuf::type,
                            micromodem::protobuf::MICROMODEM_REMUS_LBL_RANGING)
      ;

    // must call update after manipulating MACManager before calling do_work
       again.
    mac.update();
```

```
    //
    // 6. Run it
    //

    for(;;)
    {
        mac.do_work();
        usleep(100000);
    }

    return 0;
}

void init_transmission(const goby::acomms::protobuf::ModemTransmission& msg)
{
    std::cout << "starting transmission with these values: " << msg <<
      std::endl;
}
```

19.2 acomms/chat/chat.cpp

```
// usage: connect two modems and then run
// > chat /dev/tty_modem_A 1 2 log_file_A
// > chat /dev/tty_modem_B 2 1 log_file_B

// type into a window and hit enter to send a message. messages will be queued
// and sent on a fixed rotating cycle

#include <iostream>

#include "goby/acomms/dccl.h"
#include "goby/acomms/queue.h"
#include "goby/acomms/modem_driver.h"
#include "goby/acomms/amac.h"
#include "goby/acomms/bind.h"
#include "goby/util/as.h"
#include "goby/common/time.h"
#include "chat.pb.h"

#include <boost/lexical_cast.hpp>

#include "chat_curses.h"

using goby::util::as;
using goby::common::goby_time;

int startup_failure();
void received_data(const google::protobuf::Message&);
void received_ack(const goby::acomms::protobuf::ModemTransmission&,
                  const google::protobuf::Message&);
void monitor_mac(const goby::acomms::protobuf::ModemTransmission& mac_msg);

std::ofstream fout_;
goby::acomms::DCCLCodec* dccl_ =
      goby::acomms::DCCLCodec::get();
goby::acomms::QueueManager q_manager_;
goby::acomms::MMDriver mm_driver_;
goby::acomms::MACManager mac_;
ChatCurses curses_;
int my_id_;
int buddy_id_;


int main(int argc, char* argv[])
{
    //
    // Deal with command line parameters
    //

    if(argc != 5) return startup_failure();

    std::string serial_port = argv[1];

    try
    {
        my_id_ = boost::lexical_cast<int>(argv[2]);
        buddy_id_ = boost::lexical_cast<int>(argv[3]);
    }
    catch(boost::bad_lexical_cast&)
    {
        std::cerr << "bad value for my_id: " << argv[2] << " or buddy_id: " <<
```

```
  argv[3] << ". these must be unsigned integers." << std::endl;
    return startup_failure();
}

std::string log_file = argv[4];
fout_.open(log_file.c_str());
if(!fout_.is_open())
{
    std::cerr << "bad value for log_file: " << log_file << std::endl;
    return startup_failure();
}

//
//  Initialize logging
//
goby::glog.add_stream(goby::common::logger::DEBUG1, &
  fout_);
goby::glog.set_name(argv[0]);


// bind the signals of these libraries
bind(mm_driver_, q_manager_, mac_);

//
// Initiate DCCL (libdccl)
//
goby::acomms::protobuf::DCCLConfig dccl_cfg;

dccl_->validate<ChatMessage>();

//
// Initiate queue manager (libqueue)
//
goby::acomms::protobuf::QueueManagerConfig q_manager_cfg;
q_manager_cfg.set_modem_id(my_id_);
goby::acomms::protobuf::QueuedMessageEntry* q_entry = q_manager_cfg.
  add_message_entry();
q_entry->set_protobuf_name("ChatMessage");

goby::acomms::protobuf::QueuedMessageEntry::Role* src_role = q_entry->
  add_role();
src_role->set_type(goby::acomms::protobuf::QueuedMessageEntry::SOURCE_ID);
src_role->set_field("source");

goby::acomms::protobuf::QueuedMessageEntry::Role* dest_role = q_entry->
  add_role();
dest_role->set_type(
  goby::acomms::protobuf::QueuedMessageEntry::DESTINATION_ID);
dest_role->set_field("destination");

goby::acomms::connect(&q_manager_.signal_receive
  , &received_data);
goby::acomms::connect(&q_manager_.signal_ack
  , &received_ack);

//
// Initiate modem driver (libmodemdriver)
//
goby::acomms::protobuf::DriverConfig driver_cfg;
driver_cfg.set_modem_id(my_id_);
driver_cfg.set_serial_port(serial_port);

//
// Initiate medium access control (libamac)
//
goby::acomms::protobuf::MACConfig mac_cfg;
mac_cfg.set_type(goby::acomms::protobuf::MAC_FIXED_DECENTRALIZED);
mac_cfg.set_modem_id(my_id_);
goby::acomms::connect(&mac_.
  signal_initiate_transmission, &monitor_mac);

goby::acomms::protobuf::ModemTransmission my_slot;
my_slot.set_src(my_id_);
my_slot.set_dest(buddy_id_);
my_slot.set_rate(0);
my_slot.set_slot_seconds(12);
my_slot.set_type(goby::acomms::protobuf::ModemTransmission::DATA);

goby::acomms::protobuf::ModemTransmission buddy_slot;
buddy_slot.set_src(buddy_id_);
buddy_slot.set_dest(my_id_);
buddy_slot.set_rate(0);
buddy_slot.set_slot_seconds(12);
buddy_slot.set_type(goby::acomms::protobuf::ModemTransmission::DATA);

if(my_id_ < buddy_id_)
{
```

```
        mac_cfg.add_slot()->CopyFrom(my_slot);
        mac_cfg.add_slot()->CopyFrom(buddy_slot);
    }
    else
    {
        mac_cfg.add_slot()->CopyFrom(buddy_slot);
        mac_cfg.add_slot()->CopyFrom(my_slot);
    }

    //
    // Start up everything
    //
    try
    {
        dccl_->set_cfg(dccl_cfg);
        q_manager_.set_cfg(q_manager_cfg);
        mac_.startup(mac_cfg);
        mm_driver_.startup(driver_cfg);
    }
    catch(std::runtime_error& e)
    {
        std::cerr << "exception at startup: " << e.what() << std::endl;
        return startup_failure();
    }

    curses_.set_modem_id(my_id_);
    curses_.startup();

    //
    // Loop until terminated (CTRL-C)
    //
    for(;;)
    {
        std::string line;
        curses_.run_input(line);

        if(!line.empty())
        {
            ChatMessage message_out;
            message_out.set_telegram(line);

            // send this message to my buddy!
            message_out.set_destination(buddy_id_);
            message_out.set_source(my_id_);

            q_manager_.push_message(message_out);
        }

        try
        {
            mm_driver_.do_work();
            mac_.do_work();
            q_manager_.do_work();
        }
        catch(std::runtime_error& e)
        {
            curses_.cleanup();
            std::cerr << "exception while running: " << e.what() << std::endl;
            return 1;
        }
    }

    return 0;
}

int startup_failure()
{
    std::cerr << "usage: chat /dev/tty_modem my_id buddy_id log_file" <<
      std::endl;
    return 1;
}

void monitor_mac(const goby::acomms::protobuf::ModemTransmission& mac_msg)
{
    if(mac_msg.src() == my_id_)
        curses_.post_message("{control} starting send to my buddy")
      ;
    else if(mac_msg.src() == buddy_id_)
        curses_.post_message("{control} my buddy might be sending
       to me now");
}

void received_data(const google::protobuf::Message& message_in)
{
    ChatMessage typed_message_in;
    typed_message_in.CopyFrom(message_in);
    curses_.post_message(typed_message_in.source(),
```

```
      typed_message_in.telegram());
}

void received_ack(const goby::acomms::protobuf::ModemTransmission& ack_message,
                  const google::protobuf::Message& original_message)
{
    ChatMessage typed_original_message;
    typed_original_message.CopyFrom(original_message);

    curses_.post_message(
        ack_message.src(),
        std::string("{ acknowledged receiving message starting with: "
                    + typed_original_message.telegram().substr(0,5) + " }"));
}
```

19.3 acomms/dccl/dccl_simple/dccl_simple.cpp

simple.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message Simple
{
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

dccl_simple.cpp

```
// encodes/decodes a string using the DCCL codec library
// assumes prior knowledge of the message format (required fields)

#include "goby/acomms/dccl.h" // for DCCLCodec
#include "simple.pb.h" // for `Simple` protobuf message defined in simple.proto
#include "goby/util/binary.h" // for goby::util::hex_encode

using goby::acomms::operator<<;

int main()
{
    goby::acomms::DCCLCodec* dccl =
      goby::acomms::DCCLCodec::get();

    // validate the Simple protobuf message type as a valid DCCL message type
    dccl->validate<Simple>();

    // read message content (in this case from the command line)
    Simple message;
    std::cout << "input a string to send up to 10 characters: " << std::endl;
    getline(std::cin, *message.mutable_telegram());

    std::cout << "passing message to encoder:\n" << message << std::endl;

    // encode the message to a byte string
    std::string bytes;
    dccl->encode(&bytes, message);

    std::cout << "received hexadecimal string: " << goby::util::hex_encode(
      bytes) << std::endl;

    message.Clear();
    // input contents right back to decoder
    std::cout << "passed hexadecimal string to decoder: " <<
      goby::util::hex_encode(bytes) << std::endl;

    dccl->decode(bytes, &message);

    std::cout << "received message:" << message << std::endl;


    return 0;
```

```
}
```

19.4 acomms/dccl/two_message/two_message.cpp

two_message.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";
message GoToCommand
{
  option (dccl.msg).id = 125;
  option (dccl.msg).max_bytes = 32;
  required int32 destination = 1 [(dccl.field).max=31,
                                  (dccl.field).min=0,
                                  (dccl.field).precision=0];
  optional string type = 2 [(dccl.field).static_value="goto",
                            (dccl.field).codec="_static"];
  required int32 goto_x = 3 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=0];
  required int32 goto_y = 4 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=0];
  required bool lights_on = 5;
  required string new_instructions = 6 [(dccl.field).max_length=10];
  required double goto_speed = 7 [(dccl.field).max=3,
                                  (dccl.field).min=0,
                                  (dccl.field).precision=2];
}

message VehicleStatus
{
  option (dccl.msg).id = 126;
  option (dccl.msg).max_bytes = 32;
  required double nav_x = 1 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=1];
  required double nav_y = 2 [(dccl.field).max=10000,
                             (dccl.field).min=0,
                             (dccl.field).precision=1];
  required HealthEnum health = 3;
  enum HealthEnum {
    HEALTH_GOOD = 0;
    HEALTH_LOW_BATTERY = 1;
    HEALTH_ABORT = 2;
  }
}
```

two_message.cpp

```
#include "goby/acomms/dccl.h"
#include "goby/util/binary.h"
#include "two_message.pb.h"
#include <exception>
#include <iostream>

using namespace goby;
using goby::acomms::operator<<;

int main()
{
    goby::acomms::DCCLCodec& dccl = *
      goby::acomms::DCCLCodec::get();

    // validate the Simple protobuf message type as a valid DCCL message type
    dccl.validate<GoToCommand>();
    dccl.validate<VehicleStatus>();

    // show some useful information about all the loaded messages
    std::cout << dccl << std::endl;

    std::cout << "ENCODE / DECODE example:" << std::endl;
```

```
    GoToCommand command;
    command.set_destination(2);
    command.set_goto_x(423);
    command.set_goto_y(523);
    command.set_lights_on(true);
    command.set_new_instructions("make_toast");
    command.set_goto_speed(2.3456);

    VehicleStatus status;
    status.set_nav_x(234.5);
    status.set_nav_y(451.3);
    status.set_health(VehicleStatus::HEALTH_ABORT);

    std::cout << "passing values to encoder:\n"
              << command << "\n"
              << status << std::endl;

    std::string bytes2, bytes3;
    dccl.encode(&bytes2, command);
    dccl.encode(&bytes3, status);

    std::cout << "received hexadecimal string for message 2 (GoToCommand): "
              << goby::util::hex_encode(bytes2)
              << std::endl;

    std::cout << "received hexadecimal string for message 3 (VehicleStatus): "
              << goby::util::hex_encode(bytes3)
              << std::endl;

    command.Clear();
    status.Clear();

    std::cout << "passed hexadecimal string for message 2 to decoder: "
              << goby::util::hex_encode(bytes2)
              << std::endl;

    std::cout << "passed hexadecimal string for message 3 to decoder: "
              << goby::util::hex_encode(bytes3)
              << std::endl;


    dccl.decode(bytes2, &command);
    dccl.decode(bytes3, &status);

    std::cout << "received values:\n"
              << command << "\n"
              << status << std::endl;

    return 0;
}
```

19.5 acomms/modemdriver/driver_simple/driver_simple.cpp

```
//
// Usage (WHOI Micro-Modem): run
// > driver_simple /dev/tty_of_modem_A 1
//
// wait a few seconds
//
// > driver_simple /dev/tty_of_modem_B 2
//
// be careful of collisions if you start them at the same time (this is why
       libamac exists!)

// Usage (example ABCModem): run
// > driver_simple /dev/tty_of_modem_A 1 ABCDriver
// > driver_simple /dev/tty_of_modem_B 2 ABCDriver
// Also see abc_modem_simulator.cpp

#include "goby/acomms/modem_driver.h"
#include "goby/util/binary.h"
#include "goby/common/logger.h"
#include "goby/acomms/connect.h"

#include <iostream>

using goby::acomms::operator<<;


void handle_data_receive(const goby::acomms::protobuf::ModemTransmission&
      data_msg);
```

```
int main(int argc, char* argv[])
{
    if(argc < 3)
    {
        std::cout << "usage: driver_simple /dev/tty_of_modem modem_id [type:
       MMDriver (default)|ABCDriver]" << std::endl;
        return 1;
    }

    //
    // 1. Create and initialize the driver we want
    //
    goby::acomms::ModemDriverBase* driver = 0;
    goby::acomms::protobuf::DriverConfig cfg;

    // set the serial port given on the command line
    cfg.set_serial_port(argv[1]);
    using google::protobuf::uint32;
    // set the source id of this modem
    uint32 our_id = goby::util::as<uint32>(argv[2]);
    cfg.set_modem_id(our_id);

    goby::glog.set_name(argv[0]);
    goby::glog.add_stream(goby::common::logger::DEBUG2, &
      std::clog);

    if(argc == 4)
    {
        if(boost::iequals(argv[3],"ABCDriver"))
        {
            std::cout << "Starting Example driver ABCDriver" << std::endl;
            driver = new goby::acomms::ABCDriver;
        }
    }

    // default to WHOI MicroModem
    if(!driver)
    {
        std::cout << "Starting WHOI Micro-Modem MMDriver" << std::endl;
        driver = new goby::acomms::MMDriver;
        // turn data quality factor message on
        // (example of setting NVRAM configuration)
        cfg.AddExtension(micromodem::protobuf::Config::nvram_cfg, "DQF,1");
    }


    goby::acomms::connect(&driver->signal_receive
      , &handle_data_receive);

    //
    // 2. Startup the driver
    //
    driver->startup(cfg);

    //
    // 3. Initiate a transmission cycle with some data
    //

    goby::acomms::protobuf::ModemTransmission transmit_message;
    transmit_message.set_type(goby::acomms::protobuf::ModemTransmission::DATA);
    transmit_message.set_src(goby::util::as<unsigned>(our_id));
    transmit_message.set_dest(goby::acomms::BROADCAST_ID
      );
    transmit_message.set_rate(0);

    transmit_message.add_frame("Hello, world!");
    transmit_message.set_ack_requested(false);

    std::cout << transmit_message << std::endl;

    driver->handle_initiate_transmission(
      transmit_message);

    //
    // 4. Run the driver
    //

    // 10 hz is good
    int i = 0;
    while(1)
    {
        ++i;
        driver->do_work();

        // send another transmission every 60 seconds
        if(!(i % 600))
```

```
            driver->handle_initiate_transmission(
      transmit_message);

        // in here you can initiate more transmissions as you want
        usleep(100000);
    }

    delete driver;
    return 0;
}

//
// 5. Post the received data
//

void handle_data_receive(const goby::acomms::protobuf::ModemTransmission&
      data_msg)
{
    std::cout << "got a message: " << data_msg << std::endl;
}
```

19.6 acomms/queue/queue_simple/queue_simple.cpp

simple.proto

```
import "goby/acomms/protobuf/dccl_option_extensions.proto";

message Simple
{
  // see http://gobysoft.org/wiki/DcclIdTable
  option (dccl.msg).id = 124;

  // if, for example, we want to use on the WHOI Micro-Modem rate 0
  option (dccl.msg).max_bytes = 32;

  required string telegram = 1 [(dccl.field).max_length=30];
}
```

queue_simple.cpp

```
// queues a single message from the DCCL library

#include "goby/acomms/queue.h"
#include "goby/acomms/connect.h"
#include "simple.pb.h"
#include "goby/util/binary.h"
#include <iostream>


using goby::acomms::operator<<;

void received_data(const google::protobuf::Message& msg);

int main()
{
    //
    //  1. Initialize the QueueManager
    //
    goby::acomms::QueueManager q_manager;

    // our modem id (arbitrary, but must be unique in the network)
    const int our_id = 1;

    goby::acomms::protobuf::QueueManagerConfig cfg;
    cfg.set_modem_id(our_id);
    cfg.add_message_entry()->set_protobuf_name("Simple");
    q_manager.set_cfg(cfg);

    // set up the callback to handle received DCCL messages
    goby::acomms::connect(&q_manager.signal_receive
      , &received_data);

    //
    //  2. Push a message to a queue
    //
```

```
    // let's make a message to store in the queue
    Simple msg;
    msg.set_telegram("hello all!");
    q_manager.push_message(msg);

    std::cout << "1. pushing message to queue 1: " << msg << std::endl;

    // see what our QueueManager contains
    std::cout << "2. " << q_manager << std::endl;

    //
    //  3. Create a loopback to simulate the Link Layer (libmodemdriver & modem
       firmware)
    //

    std::cout << "3. executing loopback (simulating sending a message to
       ourselves over the modem link)" << std::endl;

    // pretend the modem is requesting data of up to 32 bytes

    goby::acomms::protobuf::ModemTransmission request_msg;
    request_msg.set_max_frame_bytes(32);
    request_msg.set_max_num_frames(1);

    q_manager.handle_modem_data_request(&request_msg);

    std::cout << "4. requesting data, got: " << request_msg << std::endl;
    std::cout << "\tdata as hex: " << goby::util::hex_encode(request_msg.frame(
      0)) << std::endl;

    //
    //  4. Pass the received message to the QueueManager (same as outgoing
       message)
    //
    q_manager.handle_modem_receive(request_msg);

    return 0;
}

//
//  5. Do something with the received message
//
void received_data(const google::protobuf::Message& msg)
{
    std::cout << "5. received message: " << msg << std::endl;
}
```

Index

API classes for the Dynamic Compact Control Language (includes writing custom encoders)., 79
API classes for the major components of the Goby--Acomms acoustic communications library (DCCL, Queue, AMAC, ModemDriver)., 79
add
goby::acomms::DCCLFieldCodecManager, 127
add_escape_code
goby::common::tcolor, 88
add_group
goby::common::FlexOstream, 164
add_id_codec
goby::acomms::DCCLCodec, 99
add_queue
goby::acomms::QueueManager, 158
add_value
goby::acomms::FromProtoCppTypeBase, 144
AlgFunction1
goby::transitional, 91
AlgFunction2
goby::transitional, 91
all_message_ids
goby::transitional::DCCLTransitionalCodec, 177
all_message_names
goby::transitional::DCCLTransitionalCodec, 177
any_decode
goby::acomms::DCCLFieldCodecBase, 118
any_encode
goby::acomms::DCCLFieldCodecBase, 118
any_post_decode
goby::acomms::DCCLFieldCodecBase, 118
any_pre_encode
goby::acomms::DCCLFieldCodecBase, 118
any_size
goby::acomms::DCCLFieldCodecBase, 119
Application
goby::pb::Application, 171

base_decode
goby::acomms::DCCLFieldCodecBase, 119
base_encode
goby::acomms::DCCLFieldCodecBase, 119
base_info
goby::acomms::DCCLFieldCodecBase, 119
base_max_size
goby::acomms::DCCLFieldCodecBase, 120
base_min_size
goby::acomms::DCCLFieldCodecBase, 120
base_size
goby::acomms::DCCLFieldCodecBase, 120
base_validate
goby::acomms::DCCLFieldCodecBase, 120
boost::asio::time_traits< goby::common::GobyTime >, 93

char_array2hex_string
goby::transitional, 92
ChatCurses, 94
cpp_bool
goby::transitional, 91
cpp_double
goby::transitional, 91
cpp_long
goby::transitional, 91
cpp_notype
goby::transitional, 91
cpp_string
goby::transitional, 91

DCCL_HEADER_NAMES
goby::transitional, 93
DCCLCppType
goby::transitional, 91
DCCLTransitionalCodec
goby::transitional::DCCLTransitionalCodec, 176
DCCLType
goby::transitional, 91
dccl_bool
goby::transitional, 91
dccl_enum
goby::transitional, 91
dccl_float
goby::transitional, 91
dccl_hex
goby::transitional, 91
dccl_int
goby::transitional, 91
dccl_static
goby::transitional, 91
dccl_string
goby::transitional, 91
dccl_field_options
goby::acomms::DCCLFieldCodecBase, 121
decode
goby::acomms::DCCLCodec, 99
goby::acomms::DCCLDefaultIdentifierCodec, 108
goby::acomms::DCCLDefaultNumericFieldCodec, 112
goby::acomms::DCCLRepeatedTypedFieldCodec, 132
goby::acomms::DCCLTypedFieldCodec, 138
decode_repeated
goby::acomms::DCCLCodec, 100
do_work
goby::acomms::ABCDriver, 95
goby::acomms::ModemDriverBase, 152
goby::moos::BluefinCommsDriver, 167
goby::moos::UFldDriver, 169

encode
goby::acomms::DCCLCodec, 100

goby::acomms::DCCLDefaultIdentifierCodec, 108, 109
goby::acomms::DCCLDefaultNumericFieldCodec, 112
goby::acomms::DCCLRepeatedTypedFieldCodec, 132, 133
goby::acomms::DCCLTypedFieldCodec, 138
encode_repeated
goby::acomms::DCCLCodec, 100

field_decode
goby::acomms::DCCLFieldCodecBase, 121
field_decode_repeated
goby::acomms::DCCLFieldCodecBase, 121
field_encode
goby::acomms::DCCLFieldCodecBase, 121
field_encode_repeated
goby::acomms::DCCLFieldCodecBase, 121
field_info
goby::acomms::DCCLFieldCodecBase, 122
field_max_size
goby::acomms::DCCLFieldCodecBase, 122
field_min_size
goby::acomms::DCCLFieldCodecBase, 122
field_post_decode
goby::acomms::DCCLFieldCodecBase, 122
field_post_decode_repeated
goby::acomms::DCCLFieldCodecBase, 122
field_pre_encode
goby::acomms::DCCLFieldCodecBase, 123
field_pre_encode_repeated
goby::acomms::DCCLFieldCodecBase, 123
field_size
goby::acomms::DCCLFieldCodecBase, 123
field_size_repeated
goby::acomms::DCCLFieldCodecBase, 123
field_type
goby::acomms::DCCLFieldCodecBase, 124
field_validate
goby::acomms::DCCLFieldCodecBase, 124
find
goby::acomms::DCCLFieldCodecManager, 127
flush_queue
goby::acomms::QueueManager, 158

get
goby::transitional::DCCLMessageVal, 173, 174
get_repeat
goby::transitional::DCCLTransitionalCodec, 177
get_repeated_value
goby::acomms::FromProtoCppTypeBase, 144
get_value
goby::acomms::FromProtoCppTypeBase, 144
goby, 80
run, 82
goby/moos/moos_protobuf_helpers.h, 184
goby::transitional
cpp_bool, 91
cpp_double, 91
cpp_long, 91
cpp_notype, 91
cpp_string, 91
dccl_bool, 91
dccl_enum, 91
dccl_float, 91
dccl_hex, 91
dccl_int, 91
dccl_static, 91
dccl_string, 91
goby::Exception, 166
goby::acomms, 82
goby::acomms::ABCDriver, 95
do_work, 95
handle_initiate_transmission, 95
startup, 96
goby::acomms::DCCLCodec, 96
add_id_codec, 99
decode, 99
decode_repeated, 100
encode, 100
encode_repeated, 100
id, 101
id_from_encoded, 101
info, 102
info_all, 102
load_shared_library_codecs, 102
set_id_codec, 102
size, 102
size_repeated, 103
validate, 103
validate_repeated, 104
goby::acomms::DCCLDefaultBoolCodec, 104
goby::acomms::DCCLDefaultBytesCodec, 105
goby::acomms::DCCLDefaultEnumCodec, 106
goby::acomms::DCCLDefaultIdentifierCodec, 107
decode, 108
encode, 108, 109
max_size, 109
min_size, 109
size, 109
goby::acomms::DCCLDefaultMessageCodec, 110
goby::acomms::DCCLDefaultNumericFieldCodec
decode, 112
encode, 112
goby::acomms::DCCLDefaultNumericFieldCodec< WireType, FieldType >, 111
goby::acomms::DCCLDefaultStringCodec, 113
goby::acomms::DCCLException, 114
goby::acomms::DCCLFieldCodecBase, 114
any_decode, 118
any_encode, 118
any_post_decode, 118
any_pre_encode, 118
any_size, 119
base_decode, 119
base_encode, 119
base_info, 119

base_max_size, 120
base_min_size, 120
base_size, 120
base_validate, 120
dccl_field_options, 121
field_decode, 121
field_decode_repeated, 121
field_encode, 121
field_encode_repeated, 121
field_info, 122
field_max_size, 122
field_min_size, 122
field_post_decode, 122
field_post_decode_repeated, 122
field_pre_encode, 123
field_pre_encode_repeated, 123
field_size, 123
field_size_repeated, 123
field_type, 124
field_validate, 124
info, 124
max_size, 124
min_size, 124
require, 125
this_descriptor, 125
this_field, 125
wire_type, 125
goby::acomms::DCCLFieldCodecManager, 126
add, 127
find, 127
goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, Enable >, 128
goby::acomms::DCCLRepeatedTypedFieldCodec
decode, 132
encode, 132, 133
max_size, 133
min_size, 133
size, 133, 134
goby::acomms::DCCLRepeatedTypedFieldCodec< WireType, FieldType >, 130
goby::acomms::DCCLStaticCodec< T >, 134
goby::acomms::DCCLTimeCodec< TimeType >, 135
goby::acomms::DCCLTypeHelper, 141
goby::acomms::DCCLTypedFieldCodec
decode, 138
encode, 138
size, 139
goby::acomms::DCCLTypedFieldCodec< WireType, FieldType >, 137
goby::acomms::DCCLTypedFixedFieldCodec< WireType, FieldType >, 140
goby::acomms::FromProtoCppTypeBase, 143
add_value, 144
get_repeated_value, 144
get_value, 144
set_value, 145
goby::acomms::FromProtoCustomMessage< CustomMessage >, 145
goby::acomms::FromProtoTypeBase, 146
goby::acomms::MACManager, 146
signal_initiate_transmission, 148
startup, 148
goby::acomms::MMDriver, 148
startup, 150
goby::acomms::ModemDriverBase, 150
do_work, 152
handle_initiate_transmission, 152
modem_read, 153
modem_start, 153
modem_write, 153
signal_data_request, 154
signal_modify_transmission, 154
signal_raw_incoming, 154
signal_raw_outgoing, 154
signal_receive, 154
signal_transmit_result, 155
startup, 153
goby::acomms::QueueException, 155
goby::acomms::QueueManager, 156
add_queue, 158
flush_queue, 158
handle_modem_data_request, 158
handle_modem_receive, 159
info, 159
info_all, 159
push_message, 159
signal_ack, 160
signal_data_on_demand, 160
signal_expire, 160
signal_queue_size_change, 160
signal_receive, 161
goby::common, 85
goby::common::Colors, 161
goby::common::ConfigException, 162
goby::common::FlexNCurses, 162
goby::common::FlexOStreamBuf, 164
goby::common::FlexOstream, 163
add_group, 164
goby::common::TermColor, 165
goby::common::tcolor, 88
add_escape_code, 88
goby::moos::BluefinCommsDriver, 167
do_work, 167
handle_initiate_transmission, 168
startup, 168
goby::moos::UFldDriver, 168
do_work, 169
handle_initiate_transmission, 169
startup, 170
goby::pb, 89
goby::pb::Application, 170
Application, 171
subscribe, 171
zeromq_service, 171
goby::transitional, 89
AlgFunction1, 91

AlgFunction2, 91
char_array2hex_string, 92
DCCL_HEADER_NAMES, 93
DCCLCppType, 91
DCCLType, 91
hex_string2binary_string, 92
hex_string2number, 92
number2hex_string, 92
goby::transitional::DCCLMessageVal, 172
get, 173, 174
operator bool, 174
operator double, 174
operator float, 174
operator int, 175
operator long, 175
operator std::string, 175
operator unsigned, 175
set, 175
goby::transitional::DCCLTransitionalCodec, 176
all_message_ids, 177
all_message_names, 177
DCCLTransitionalCodec, 176
get_repeat, 177
id2name, 177
message_count, 177
message_var_names, 177
name2id, 177
goby::util::LineBasedInterface, 178
readline, 179
goby::util::SerialClient, 179
SerialClient, 180
goby::util::TCPClient, 180
TCPClient, 181
goby::util::TCPServer, 181
TCPServer, 182
Group, 183
GroupSetter, 183

handle_initiate_transmission
goby::acomms::ABCDriver, 95
goby::acomms::ModemDriverBase, 152
goby::moos::BluefinCommsDriver, 168
goby::moos::UFldDriver, 169
handle_modem_data_request
goby::acomms::QueueManager, 158
handle_modem_receive
goby::acomms::QueueManager, 159
hex_string2binary_string
goby::transitional, 92
hex_string2number
goby::transitional, 92

id
goby::acomms::DCCLCodec, 101
id2name
goby::transitional::DCCLTransitionalCodec, 177
id_from_encoded
goby::acomms::DCCLCodec, 101
info
goby::acomms::DCCLCodec, 102
goby::acomms::DCCLFieldCodecBase, 124
goby::acomms::QueueManager, 159
info_all
goby::acomms::DCCLCodec, 102
goby::acomms::QueueManager, 159

load_shared_library_codecs
goby::acomms::DCCLCodec, 102

max_size
goby::acomms::DCCLDefaultIdentifierCodec, 109
goby::acomms::DCCLFieldCodecBase, 124
goby::acomms::DCCLRepeatedTypedFieldCodec, 133
message_count
goby::transitional::DCCLTransitionalCodec, 177
message_var_names
goby::transitional::DCCLTransitionalCodec, 177
min_size
goby::acomms::DCCLDefaultIdentifierCodec, 109
goby::acomms::DCCLFieldCodecBase, 124
goby::acomms::DCCLRepeatedTypedFieldCodec, 133
modem_read
goby::acomms::ModemDriverBase, 153
modem_start
goby::acomms::ModemDriverBase, 153
modem_write
goby::acomms::ModemDriverBase, 153
moos_protobuf_helpers.h
parse_for_moos, 185
serialize_for_moos, 185

name2id
goby::transitional::DCCLTransitionalCodec, 177
number2hex_string
goby::transitional, 92

operator bool
goby::transitional::DCCLMessageVal, 174
operator double
goby::transitional::DCCLMessageVal, 174
operator float
goby::transitional::DCCLMessageVal, 174
operator int
goby::transitional::DCCLMessageVal, 175
operator long
goby::transitional::DCCLMessageVal, 175
operator std::string
goby::transitional::DCCLMessageVal, 175
operator unsigned
goby::transitional::DCCLMessageVal, 175

parse_for_moos
moos_protobuf_helpers.h, 185
post_decode
goby::acomms::DCCLFieldCodecSelector< WireType, FieldType, typename boost::disable_if<

boost::is_same< WireType, FieldType > >-
::type >, 129
pre_encode
goby::acomms::DCCLFieldCodecSelector< Wire-
Type, FieldType, typename boost::disable_if<
boost::is_same< WireType, FieldType > >-
::type >, 130
push_message
goby::acomms::QueueManager, 159

readline
goby::util::LineBasedInterface, 179
require
goby::acomms::DCCLFieldCodecBase, 125
run
goby, 82

SerialClient
goby::util::SerialClient, 180
serialize_for_moos
moos_protobuf_helpers.h, 185
set
goby::transitional::DCCLMessageVal, 175
set_id_codec
goby::acomms::DCCLCodec, 102
set_value
goby::acomms::FromProtoCppTypeBase, 145
signal_ack
goby::acomms::QueueManager, 160
signal_data_on_demand
goby::acomms::QueueManager, 160
signal_data_request
goby::acomms::ModemDriverBase, 154
signal_expire
goby::acomms::QueueManager, 160
signal_initiate_transmission
goby::acomms::MACManager, 148
signal_modify_transmission
goby::acomms::ModemDriverBase, 154
signal_queue_size_change
goby::acomms::QueueManager, 160
signal_raw_incoming
goby::acomms::ModemDriverBase, 154
signal_raw_outgoing
goby::acomms::ModemDriverBase, 154
signal_receive
goby::acomms::ModemDriverBase, 154
goby::acomms::QueueManager, 161
signal_transmit_result
goby::acomms::ModemDriverBase, 155
size
goby::acomms::DCCLCodec, 102
goby::acomms::DCCLDefaultIdentifierCodec, 109
goby::acomms::DCCLRepeatedTypedFieldCodec,
133, 134
goby::acomms::DCCLTypedFieldCodec, 139
size_repeated
goby::acomms::DCCLCodec, 103
startup
goby::acomms::ABCDriver, 96
goby::acomms::MACManager, 148
goby::acomms::MMDriver, 150
goby::acomms::ModemDriverBase, 153
goby::moos::BluefinCommsDriver, 168
goby::moos::UFldDriver, 170
subscribe
goby::pb::Application, 171

TCPClient
goby::util::TCPClient, 181
TCPServer
goby::util::TCPServer, 182
this_descriptor
goby::acomms::DCCLFieldCodecBase, 125
this_field
goby::acomms::DCCLFieldCodecBase, 125

validate
goby::acomms::DCCLCodec, 103
validate_repeated
goby::acomms::DCCLCodec, 104

wire_type
goby::acomms::DCCLFieldCodecBase, 125

zeromq_service
goby::pb::Application, 171